N° d'ordre :

THÈSE

présentée à

L'UNIVERSITÉ PIERRE ET MARIE CURIE
PARIS VI

pour obtenir

LE DIPLOME DE DOCTEUR DE 3ème CYCLE

par

Patrick VALDURIEZ

Spécialité : **Mathématiques**
Mention : **Informatique**

Sujet de la thèse :

UN PROCESSEUR DE JOINTURE ET TRI POUR UNE MACHINE BASES DE DONNÉES

Soutenue le 16 décembre 1981 devant la Commission composée de :

MM.	**C.**	**GIRAULT**	Président
	M.	**ADIBA**	Examinateurs
	F.	**BANCILHON**	
	G.	**GARDARIN**	
	J.	**LE BIHAN**	

UNIV.
LUZERN
BIBL.

N° d'ordre :

THÈSE

présentée à

L'UNIVERSITÉ PIERRE ET MARIE CURIE
PARIS VI

pour obtenir

LE DIPLOME DE DOCTEUR DE 3ème CYCLE

par

Patrick VALDURIEZ

Spécialité : **Mathématiques**
Mention : **Informatique**

Sujet de la thèse :

UN PROCESSEUR DE JOINTURE ET TRI POUR UNE MACHINE BASES DE DONNÉES

Soutenue le 16 décembre 1981 devant la Commission composée de :

MM.	C.	**GIRAULT**	Président
	M.	**ADIBA**	Examinateurs
	F.	**BANCILHON**	
	G.	**GARDARIN**	
	J.	**LE BIHAN**	

Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41547321_0003

Je remercie vivement :

. *Claude GIRAULT, qui m'a fait l'honneur de présider mon Jury,*

. *Georges GARDARIN, qui m'a initié à la recherche en m'accueillant dans son équipe et qui m'a dirigé dans mon travail,*

. *Michel ADIBA, pour ses commentaires et conseils fructueux,*

. *François BANCILHON, pour m'avoir fait l'honneur de participer à mon Jury,*

. *Jean LE BIHAN, Chef du Projet SIRIUS, projet dans lequel ce travail a commencé.*

Je voudrais également exprimer toute ma reconnaissance aux membres du projet SABRE : Z. BELLAHSENE, P. BERNADAT, P. FAUDEMAY, K. KARLSSON, G. KUBRYK, N. TEMMERMAN, M. THONNET et Y. VIEMONT.

Enfin, je remercie vivement tous ceux qui m'ont soutenu tout au long de l'élaboration de cette Thèse, particulièrement,
Cati.

AVERTISSEMENT

Dans cette thèse, nous proposons des algorithmes pour réaliser les opérations de jointure et tri dans une machine bases de données multiprocesseur. La présentation de ce travail a fait l'objet d'un effort particulier afin de faciliter sa compréhension et de permettre sa lecture partielle. Ainsi, tous les chapitres, exepté le dernier, peuvent être lus indépendamment des autres.

L'introduction résume le contenu de la thèse et les principaux résultats obtenus. Le chapitre I décrit le contexte d'étude, la machine bases de données SABRE. Le chapitre II présente l'état de l'art et fait une analyse critique des solutions connues. Le chapitre III propose des algorithmes multiprocesseurs de jointure et tri de relations. Le chapitre IV introduit l'intérêt de l'opérateur de semi-jointure et propose des algorithmes pour systèmes multiprocesseurs. Le chapitre V décrit finalement les spécifications d'un processeur de jointure et tri pour une machine bases de données.

Tous les chapitres peuvent être lus isolément à l'exeption du chapitre V qui nécéssite la lecture des chapitres III et IV.

TABLE DES MATIERES

INTRODUCTION

1. PROBLEME ABORDE 6
2. ETAT DE L'ART 10
3. RESULTATS DE RECHERCHE 11
4. PLAN DE THESE 13

CHAPITRE I : CONTEXTE DE L'ETUDE : SABRE

1. INTRODUCTION 18
2. OBJECTIFS DE SABRE 20
3. PRINCIPES DE BASE 25
4. ARCHITECTURE DE SABRE 28
5. CONCLUSION 36

CHAPITRE II : ANALYSE DE DIVERSES METHODES DE JOINTURE DE RELATIONS

1. INTRODUCTION 40
2. NOTATIONS 42
3. CAS SANS CHEMINS D'ACCES 43
4. CAS AVEC INDEX SECONDAIRES 51
5. CAS AVEC CLES 58
6. COMPARAISONS 60
7. CONCLUSION 67

CHAPITRE III : ALGORITHMES DE JOINTURE MULTIPROCESSEURS

1. INTRODUCTION 70

2. MODELE D'ARCHITECTURE 72

3. PARAMETRES ET CRITERES D'ANALYSE 75

4. JOINTURE PAR PRODUIT DES RELATIONS 79

5. JOINTURE PAR TRI DES RELATIONS 81

6. JOINTURE PAR HACHAGE D'UNE RELATION 87

7. COMPARAISONS 91

8. CONCLUSION 99

CHAPITRE IV : ALGORITHMES DE SEMI-JOINTURE DE RELATIONS POUR SYSTEMES MULTIPROCESSEURS

1. INTRODUCTION 102

2. MODELE D'ARCHITECTURE 104

3. PARAMETRES ET CRITERES D'ANALYSE 107

4. ALGORITHMES D'EQUI-SEMI-JOINTURE 110

5. ALGORITHMES D'INEQUI-SEMI-JOINTURE 121

6. JOINTURES UTILISANT DES SEMI-JOINTURES 122

7. CONCLUSION 129

CHAPITRE V : SPECIFICATIONS D'UN PROCESSEUR
DE JOINTURE ET TRI

1. INTRODUCTION 132
2. FONCTIONS DES PROCESSEURS DE JOINTURE ET TRI 133
3. ALLOCATION DES PROCESSEURS DE JOINTURE ET TRI 137
4. SPECIFICATIONS D'UN PROCESSEUR DE JOINTURE ET TRI 139
5. SERVICES DEMANDES 148
6. CONCLUSION 153

CONCLUSION 156

REFERENCES 161

ANNEXES

ANNEXE 1 : PARAMETRES MATERIELS 170
ANNEXE 2 : JEUX D'ESSAI DU PJT 171

INTRODUCTION

1. PROBLEME ABORDE

L'intérêt porté au modèle relationnel (CODD70) par les concepteurs de systèmes de gestion de bases de données est dû à sa qualité essentielle d'autoriser des langages de manipulation de données assertionnels. L'utilisateur précise une information sur le contenu des données (quoi) plutôt que la manière de les accéder (comment). Ainsi, une meilleure indépendance entre la manipulation de données par l'utilisateur et les chemins d'accès à ces données est offerte. Pour obtenir de bonnes performances des systèmes relationnels, la gestion et l'optimisation des chemins d'accès deviennent des objectifs primordiaux. Cependant, le développement de tels systèmes pour gérer de grandes bases de données est limité par plusieurs facteurs : complexité de traduction de noms, faiblesse relative des vitesses d'entrées-sorties, goulots d'étranglement. Afin de diminuer le nombre d'accès aux mémoires secondaires et retrouver rapidement les données, des structures de données complexes (index, liens, etc) sont exploitées. La mise à jour de ces informations auxiliaires rend très coûteuse en temps d'exécution les opérations de modifications. D'autre part, les différents modules réalisant les fonctions de gestion de bases de données exigent des possibilités de grandes performances et font l'objet d'optimisations diverses. L'implantation de ces modules sur le même matériel rend difficile de telles optimisations et provoque de nombreux goulots d'étranglement (par exemple, les entrées-sorties disques).

Un facteur limitatif essentiel aux performances des systèmes de gestion de bases de données relationnels est l'opération de jointure (appelée aussi produit, jonction ou composition). La θ-jointure (CODD70) de deux relations R et S selon les attributs A de R et B de S est la relation résultat T obtenu en concaténant tout tuple de R avec tout tuple de S, tels que A θ B, ou θ est un des opérateurs $=, <, \leq, \geq, >, \neq$.

exemple de jointure

Soit la base de données composée de deux relations EMPLOYE (nom, salaire, service) et SERVICE (service, nom chef, salaire) , décrite figure 1.

EMPLOYE

nom	salaire	service
DUPONT	5000	VENTE
DURAND	4000	ACHAT
DUPUY	3000	VENTE
DUMONT	5000	TRI

SERVICE

service	chef	salaire
ACHAT	PIERRE	6000
VENTE	JEAN	6200
LABO	PAUL	5000

figure 1 : exemple de base de données

La jointure des relations EMPLOYE et SERVICE sur les attributs service avec l'opérateur = (equi-jointure) est la relation EMPSER décrite figure 2. La deuxième colonne de l'attribut service , redondante, a été supprimée et l'opération est appelée jointure naturelle.

EMPSER

nom	salaire	service	chef	salaire
DUPONT	5000	VENTE	JEAN	6200
DURAND	4000	ACHAT	PIERRE	6000
DUPUY	3000	VENTE	JEAN	6200

figure 2 : relation résultant de la jointure

Cette opération de base est très fréquente car elle est nécessaire au traitement de toute question portant sur plusieurs relations. De plus, elle peut être très coûteuse en temps d'exécution. Joindre deux relations équivaut à faire l'intersection de deux ensembles , où chaque élément du premier ensemble est comparé à chaque élément du second. Sans chemin d'accès particuliers, ceci peut conduire pour des ensembles de n éléments, à un coût proportionnel à (n**2). Si une entrée-sortie est nécessaire pour accéder à un élément, ce coût est prohibitif lorsque

n est grand.

Pour résoudre les problèmes de gestion de grandes bases de données, de récentes solutions ont été proposées. Elles consistent à isoler les fonctions de gestion de bases de données du calculateur hôte et les regrouper dans une machine spécialisée, appelée machine bases de données. Cependant la plupart d'entre elles nécessitent le développement d'un matériel particulier, qui les révèle souvent coûteuses. Généralement, l'architecture de ces machines est orientée pour supporter un type particulier d'applications.

Une première approche à la solution machines bases de données est appelée "disques associatifs", dont les systèmes CASSM (LIPO75), RARES (LIN76), RAP (OZKA75), et dans une certaine limite, DBC (BANE78), sont issus. Le but de cette approche est l'exécution efficace de l'opération de restriction, permettant de retrouver rapidement les données qui satisfont un certain critère. Pour ce faire, la base de données est stockée sur les pistes d'un disque associé à une capacité de traitement suffisante pour permettre l'exécution des recherches en parallèle. Etant donné cette architecture, les autres opérations relationnelles telles que jointure ou projection, sont réalisées avec plus ou moins de succès, en utilisant le calculateur hôte et les opérations implantées dans la machine bases de données.

Une seconde approche, appelée "processeurs associatifs", consiste à stocker une partie de la base sur mémoire secondaire et une partie en mémoire accédée par des processeurs parallèles. RAP (OZKA75) et STARAN (DEFI73) en sont des exemples. La capacité limitée et la nécessité d'un matériel hautement performant et coûteux font que cette approche n'est pas viable.

Une troisième approche, qualifiée de "processeur par disque" ,

consiste à placer un processeur entre le disque, où est stockée la base de données, et la mémoire principale. Ce processeur, appelé filtre (BANC80), sélectionne les tuples qui satisfont un critère de restriction vers la mémoire principale. L'avantage de cette approche est de fournir la même fonction de sélection et d'être faisable avec la technologie existante. Cependant, les performances sont plus faibles que celles des " disques associatifs" (DEWI81).

Une dernière approche, plus récente, qualifiée de "multiprocesseur", utilise un ensemble de processeurs et une grande mémoire cache, communiquant au moyen d'un réseau d'interconnexion. DIRECT (DEWI79) est un exemple de projet opérationnel. L'objectif majeur qui a motivé ce type d'architecture est d'obtenir des performances proportionnelles au nombre de processeurs mis en oeuvre.

Le contexte de notre étude est le projet SABRE. Il est dérivé du couplage des deux dernières approches, "processeur par disque" et "multiprocesseur". Ce projet a pour ambition d'offrir des gains appréciables en performances et fonctionnalité, en utilisant des technologies existantes. SABRE gère des bases de données relationnelles, manipulées au moyen d'un langage assertionnel. L'architecture est caractérisée par une organisation multiprocesseur offrant un haut degré de parallélisme. Les fonctions de gestion de bases de données sont réparties entre les différents processeurs. Cet aspect multiprocesseur permet deux types de parallélisme : le parallélisme inter-requête qui consiste a traiter simultanément des requêtes de différents utilisateurs, et le parallélisme intra-requête permettant de traiter en parallèle une même requête coûteuse.

Le sujet de cette thèse est d'étudier et de proposer, dans un tel contexte, des méthodes multiprocesseurs pour réaliser les opérations coûteuses. Ces opérations exigent un traitement répétitif sur de

grandes relations qui ne peuvent être contenues en mémoire, et, à chaque étape de traitement nécessitent une lecture et (ou) écriture de la (les) relation(s). Ces opérations sont la jointure, le tri externe et l'élimination des tuples dupliqués dans une relation. L'objectif est d'atteindre un coût d'exécution inversement proportionnel au nombre de processeurs traitants, en profitant au mieux du parallélisme inhérent à SABRE.

2. ETAT DE L'ART

Le chapitre II de cette thèse est consacré a l'étude critique et l'analyse des méthodes classiques pour réaliser l'opération relationnelle de jointure. Il ressort de ce chapitre trois groupes de méthodes permettant d'éviter le produit cartésien des relations. Le premier groupe, basé sur le tri, permet d'obtenir des temps d'exécution de l'ordre de n*log n, pour des relations de n tuples. Le deuxième, basé sur le hachage, donne de meilleurs résultats de l'ordre de n, mais est seulement utilisable pour la semi-jointure, cas particulier de jointure ou les attributs résultats n'appartiennent qu'à une seule relation. Le troisième groupe utilise des index secondaires sur les attributs de jointure et les performances dépendent essentiellement du placement des tuples sur mémoire secondaire et du nombre de tuples résultats. Ce dernier groupe est utile pour les cas de jointure fréquents mais peut être très inefficace, si les tuples ne sont pas placés par groupement sur l'attribut de jointure. Il est aussi à noter que ces méthodes nécessitent un coût supplémentaire de mise à jour des index.

Dans ce chapitre traitant de l'état de l'art, nous proposons une méthode de jointure simple en combinant la méthode par hachage (semi-jointure) avec le tri. Elle se révèle être généralement la plus

efficace. En général, il s'avère que les méthodes de jointure des systèmes classiques sont adaptées à un certain type de question, invoquant généralement des opérations de restriction-projection puis jointure.

3. RESULTATS DE RECHERCHE

Nous proposons différentes solutions pour améliorer les performances des opérateurs généralement coûteux lorsque les tailles des opérandes sont élevées. L'opérateur de jointure est le plus coûteux et le plus difficile à maitriser dans un contexte multiprocesseur. L'essentiel de cette thèse y est consacré et constitue notre principale contribution à la recherche. Trois algorithmes de jointure, inspirés des méthodes monoprocesseurs sans index, et adaptés à un contexte multiprocesseur sont présentés. Le premier algorithme est une version parallèle de la méthode monoprocesseur la plus inefficace. Elle consiste à répartir une relation (externe) sur un ensemble de processeurs, puis à diffuser l'autre relation (interne) de sorte que chaque processeur joigne une partie de la relation externe à la relation interne entière. Le deuxième algorithme réalise le tri ascendant parallèle des relations sur l'attribut de jointure et compose la jointure dans une dernière étape de fusion. Nous proposons là un nouvel algorithme de tri multiprocesseur. Le troisième algorithme consiste à générer un fichier haché sur l'attribut de jointure, à partir d'une relation, puis à répartir la deuxième sur un ensemble de processeurs et, pour chaque tuple suceptible d'appartenir à la jointure, accéder directement les tuples du fichier haché. Cet algorithme est optimisé grâce à des tableaux de bits. L'analyse des coûts d'exécution de ces algorithmes, que nous avons réalisé, montre que chacun d'eux possède un créneau d'application défini par les caractéristiques des relations à joindre et du nombre de processeurs. Le premier algorithme, par

produit, est le plus simple. Il est également celui qui se comporte le mieux quand le nombre de processeurs exécutant la jointure en parallèle devient grand. L'algorithme par tri, plus complexe, est performant relativement aux autres pour des tailles de relations a joindre élevées. Cet algorithme a l'avantage de produire la relation résultat triée sur l'attribut de jointure, et aussi d'utiliser un opérateur nécéssaire à la machine (tri). Le dernier algorithme, par hachage, est meilleur lorsque la proportion de tuples de la plus grande relation participant à la jointure n'est pas très forte. Dans ce cas, le dispositif tableau de bits permet d'éviter des accès inutiles au fichier haché. Ce sont là des résultats importants qui ont conduit à la conception de la machine bases de données SABRE.

La semi-jointure est aussi une opération importante à prendre en compte puisqu'elle est un cas particulier de jointure. Elle a la propriété de restreindre le nombre de tuples des relations à joindre. Nous proposons encore deux algorithmes d'equi-semi-jointure et un algorithme d'inequi-semi-jointure. Les temps d'excution sont linéairement proportionnels au volume des opérandes et du résultat et inversement proportionnels au nombre de processeurs traitants. La comparaison entre la méthode de jointure de relations et la méthode de jointure de leurs semi-jointures est effectuée. Nous montrons que la seconde est généralement la meilleure. Ceci montre l'intérêt à réaliser l'opération de semi-jointure dans une machine bases de données.

En conclusion, nous proposons plusieurs solutions à un problème plutôt qu'une solution, qui ne peut répondre bien à tous les cas. La meilleure solution sera choisie en fonction de la question et d'informations statistiques sur les relations. Ces informations statistiques sont aussi utilisées lors de l'optimisation de question. Nous remarquons aussi que les solutions proposées peuvent être appliqués

en utilisant des technologies existantes (micro-processeurs généraux).

4. PLAN DE THESE

4-1. La machine bases de données SABRE

SABRE est le contexte de l'étude. Dans ce chapitre, les objectifs et les concepts de base de SABRE sont introduits. SABRE est une machine bases de données relationnelles, dont l'objectif principal est d'offrir des gains appréciables en performances et fonctionnalité. Les concepts de base sont le parallélisme, le filtrage des données et le placement des données sur mémoires secondaires par regroupement multi-attribut. SABRE est organisé comme un ensemble de (micro)processeurs généraux possédant une mémoire locale, et une grande mémoire servant de cache aux disques. Les processeurs communiquent au moyen d'un réseau d'interconnexion.

4-2. Analyse de diverses méthodes de jointure de relations

Ce chapitre traite de l'état de l'art concernant les algorithmes de jointure. Une synthèse et une comparaison des principales méthodes proposées pour traiter efficacement l'opération relationnelle de jointure sont faites. Des algorithmes implantés dans des systèmes existants ou seulement décrits sont étudiés et évalués en tenant compte des coûts d'entrées-sorties et du coût de stockage, considérés comme critères de performances critiques des systèmes classiques. Outre l'aspect synthèse, ce chapitre apporte une comparaison rigoureuse des algorithmes en fonction de divers paramètres : taille des relations, sélectivité des attributs, présence d'index, attributs de jointure clés.

4-3. Algorithmes multiprocesseurs de jointure de relations

Ce chapitre présente et analyse trois algorithmes pour traiter

l'opération de jointure de relations dans la machine bases de données SABRE. D'abord, le modèle d'architecture multiprocesseur est décrit. Les critères d'analyse des algorithmes incluent le coût d'entrée-sortie, le coût de traitement CPU, et le coût de transmission de messages. Puis trois algorithmes de jointure sont présentés : l'algorithme par produit, l'algorithme par tri et l'algorithme par hachage. Les comparaisons de ces algorithmes en temps d'exécution montrent que l'algorithme par produit est performant avec un grand nombre de processeurs, que l'algorithme par tri se comporte le mieux pour de grandes relations et que l'algorithme par hachage est efficace lorsque la relation résultat comporte peu de tuples.

4-4. Algorithmes de semi-jointure de relations pour systèmes multiprocesseurs

L'opérateur de semi-jointure procède en ne retenant de la base que les données utiles à la question et diminue ainsi le volume des relations opérandes. Ce chapitre présente et analyse des algorithmes pour réaliser les semi-jointures dans SABRE. Comme au chapitre précédent, le modèle d'architecture et les critères d'analyse sont décrits. Puis deux algorithmes d'equi-semi-jointure et un algorithme d'inequi-semi-jointure sont présentés. Les coûts d'exécution de ces algorithmes sont généralement proportionnels à la taille des relations opérandes et résultats et inversement proportionnels au nombre de processeurs. Puis, la méthode de jointure de deux relations et la méthode de jointure de leurs semi-jointures sont comparées. Il est finalement montré que la méthode par semi-jointures est généralement meilleure.

4-5. Spécifications d'un processeur de jointure et tri

Ce dernier chapitre décrit finalement les spécifications d'une classe de processeurs spécialisés, assignés aux quatre fonctions de base

: jointure, tri, semi-jointure, et élimination des doubles après projection. Les algorithmes supportés par chaque processeur de jointure et tri sont rappelés, et la stratégie d'allocation des processeurs a une opération parallèle est introduite. Les services rendus par ces processeurs sont décrits sous forme de primitives exprimées en PASCAL, langage qui permettra la validation des interfaces entre les différentes classes de processeurs. Les services demandés par ces processeurs sont alors présentés par une liste de primitives PASCAL. Les jeux de tests du processeur de jointure et tri de la version SABRE monoprocesseur sont donnés en annexe 2.

4-6. Conclusion

Finalement, le bilan des résultats de cette thèse est présenté en conclusion. D'autre part, l'extension de ces résultats et les directions possibles de future recherche en ce domaine sont évoquées.

CHAPITRE I

CONTEXTE DE L'ETUDE

LA MACHINE BASES DE DONNEES SABRE

1. INTRODUCTION

Pour résoudre les problèmes de gestion de grandes bases de données qui limitent les performances des systèmes classiques (ASTR76, STON76), de récentes solutions ont été proposées. Elles consistent à isoler les fonctions de gestion de données du calculateur hôte et les regrouper dans une machine spécialisée, appelée machine bases de données. Cependant, la plupart de ces machines en cours de développement (BABB79, BANE78, BRAT80, LIPO75, OZKA75) nécessitent la conception d'un matériel particulier qui les révèle très coûteuse. De plus, certaines fonctions telles que l'intégrité et la sécurité des données restent difficiles à maitriser.

Le projet SABRE (GARD81) a pour ambition la conception d'une machine bases de données offrant des gains appréciables en performances et fonctionnalité, en utilisant les technologies existantes et nouvelles (multi-microprocesseur, VLSI, mémoires a bulles etc). SABRE a pour objectif la gestion de bases de données relationnelles manipulées au moyen d'un langage assertionnel de haut niveau, qui permet à l'utilisateur de poser des questions complexes. La puissance du langage d'interrogation conditionne le succès de cette machine par rapport à un système de gestion de bases de données classique. En effet, une machine n'autorisant que des questions simples serait moins performante qu'un système de gestion de base de données classique. Ceci serait du au coût supplémentaire de communication entre le calculateur hôte et la machine bases de données.

L'architecture de SABRE est caractérisée par une organisation multiprocesseur offrant un haut degré de parallélisme. Finalement, SABRE est connectable à un calculateur hôte ou à un réseau local ou général, et dans ce dernier cas, peut s'intégrer aisément comme site de stockage d'un système de gestion de base de données réparties, tel que

SIRIUS-DELTA (LEBI80).

Ce chapitre présente le projet SABRE, développé a l'INRIA, comme contexte de l'étude. Tout d'abord, sont décrits les principaux objectifs de SABRE : permettre la gestion de bases de données réparties, offrir un langage de manipulation de données de haut niveau, favoriser le parallélisme pour améliorer les performances, diminuer les temps d'entrées-sorties, accélérer les recherches de données par des chemins d'accès efficaces, et finalement maintenir l'intégrité et la sécurité des données.

Ensuite, les concepts de base de SABRE seront introduit au paragraphe 3 : partitions associatives, filtrage "a la volée", groupement multi-dimensionnel, opérations parallèles, décomposition de question, protocole de manipulation de données, noyau d'exécution réparti.

Finalement, au paragraphe 4, l'architecture de SABRE est présentée des points de vue fonctionnel et opérationnel. L'architecture fonctionnelle comporte 5 niveaux de fonctions correspondant par des interfaces, où chaque niveau manipule un granule d'information de plus en plus fin : base de données répartie, vue, base de données locale, relation, partition. L'architecture fonctionnelle se traduit en architecture opérationnelle par l'affectation de processeurs à chaque fonction, l'ensemble des processeurs communiquant au moyen d'un réseau d'interconnexion. Le contrôle des processeurs et des synchronisations est réalisé par un noyau d'exécution réparti, qui sera implanté par une couche de logiciel sur chaque processeur.

2. OBJECTIFS DE SABRE

2-1. Gestion de bases de données réparties

La machine bases de données SABRE est d'abord un outil facilitant la gestion de bases de données réparties, en s'intégrant dans un système de gestion de base de données réparties (LEBI80). Un tel système conduit à distinguer deux types de sites (GARD79a) : sites d'exécution de transactions, ou l'utilisateur interroge la base de données répartie, et sites de stockage des données. SABRE peut intervenir comme site de stockage et gestion de ces données et, dans ce cas, offrir les fonctions nécessaires à son intégration dans un système réparti (ADIB78). Les fonctions essentielles incluent le traitement de questions portant sur des données réparties (BERN80), la préservation de la cohérence des données dupliquées (CHU80), et la résistance aux pannes qui peut être réalisé, par exemple, en respectant un protocole de "commit" en deux étapes des transactions (BAER80).

2-2. Langage de manipulation de données assertionnel

Afin d'assurer l'indépendance des questions par rapport aux données et ainsi de permettre l'optimisation des accès aux données, SABRE offre aux utilisateurs des commandes permettant de définir l'ensemble des données à accéder (quoi) sans avoir a préciser la manière de les accéder (comment). L'interface externe est donc un langage de manipulation de données assertionnel pour interroger et mettre à jour des bases de données relationnelles (STON76). Une base de données apparait alors comme un ensemble de relations permanentes. Depuis ces relations permanentes, il est possible de définir des relations temporaires par combinaison d'attributs ou (et) selection de tuples. La durée de vie d'une relation temporaire est celle de la transaction qui la crée. Une transaction peut sélectionner des données dans des relations permanentes

dans une ou plusieurs relation temporaires. Une expression de sélection est composée d'une suite de références d'attributs à retenir (projection) et d'une condition de sélection. Une condition de sélection est une conjonction (et) de disjonctions (ou) de clauses de sélection. Une clause de sélection est de la forme (v1.A1 Θ v2.A2), ou v1 et v2 sont des variables représentant des relations, A1 et A2 des attributs et Θ un opérateur de comparaison $<,\leq,=,\neq,\geq,>$. Une clause de sélection plus simple ne portant que sur une relation est de la forme (v1.A1 Θ C), ou C est une constante. Une transaction peut aussi lire, insérer et mettre à jour des tuples depuis (dans) une relation temporaire. A la fin de la transaction, les relations temporaires mises à jour sont intégrées aux relations permanentes si la transaction s'est correctement exécutée. Ceci permet l'isolation des erreurs de transactions de la base permanente.

Finalement, le chargement d'une base de données dans SABRE est très simple. La description de la base s'effectue par insertion et modification de tuples dans une méta-base, qui contient le schéma de la base considérée. Ceci évite l'utilisation d'un langage de description de données particulier. La méta-base comporte deux relations par base gérée : relation et attribut. Une première version du langage est présentée dans (GARD80b).

2-3. Parallélisme

SABRE est réalisé à l'aide d'un ensemble de processeurs. Les fonctions de la machine bases de données sont réparties entre les différents processeurs. Chaque processeur réalise une fonction particulière (BANE79). Une même fonction est généralement dupliquée sur plusieurs processeurs, qui se partagent ainsi la charge due à cette fonction et qui peuvent de plus se remplacer en cas de panne. Cet aspect multiprocesseur permet deux types de parallélisme. Le premier

est le parallélisme inter-requêtes et consiste à traiter simultanément des requêtes de différents utilisateurs. Le second est le parallélisme intra-requête qui permet de traiter une même requête coûteuse par plusieurs processeurs.

2-4. Diminution du temps d'entrée-sortie

Le goulot d'étranglement essentiel des systèmes de gestion de grandes bases de données est du aux entrées-sorties disques. Afin de réduire ce temps d'entrée-sortie, SABRE met en oeuvre deux moyens :

(1) Une grande mémoire RAM (quelques millions d'octets) sert de cache devant les disques et permet de conserver les résultats intermédiaires et finals des manipulations de données. Cette mémoire est gérée par un ou plusieurs processeurs comme une mémoire virtuelle.

(2) Les données sont filtrées en parallèle à leur transferts disques vers la mémoire cache, ou les sélections sont effectuées "a la volée" par des processeurs associés chacun à une unité de disque (BANC80, FAUD80). Ainsi, seules les données utiles à la réponse de l'utilisateur sont dirigées vers la mémoire cache. De plus, l'opération de sélection est simultanée à l'entrée-sortie disque récupérant ainsi le temps de transfert.

2-5. Accélération des recherches

Afin de réduire le nombre d'entrées-sorties nécessaires pour balayer une grande relation, il est important de développer l'accélération des recherches, c'est à dire gérer des chemins d'accès aux données. Trois types de chemins d'accès sont proposés :

(1) arbres de recherche multi-dimensionnels afin de réaliser un placement des données par groupement multi-attribut.

(2) index classiques et fonctions de hachage.

(3) tables liant différentes relations pour mémoriser des jointures pré-évaluées.

La réalisation physique de ces chemins d'accès est stockée dans des relations particulières. Ceci permet de rendre le stockage optimal et d'avoir des chemins d'accès optionnels.

2-6. Intégrité

Une base de données est dans un état cohérent si elle vérifie un ensemble d'assertions appelé contraintes d'intégrité. La cohérence d'une base de données peut être altérée par trois facteurs (GARD80c) :

(1) les mises à jour incorrectes de l'utilisateur,

(2) les problèmes de concurrence entre transactions exécutées en parallèle,

(3) les pannes de type matériel ou logiciel.

SABRE met en oeuvre des outils pour maitriser ces trois facteurs. Premièrement, les mises à jour érronées seront écartées, en vérifiant un ensemble prédéfini de contraintes d'intégrité associées à chaque base de données (GARD79b). Deuxièmement, pour éviter les séquences d'opérations impropres conduisant à des mises à jour incohérentes, l'exécution simultanée des transactions doit vérifier la propriété de sérialisabilité (GARD77). Pour ce faire, deux méthodes de contrôle de concurrence sont offertes à l'administrateur de la base de données, qui lui permet ainsi de mieux adapter la solution choisie à l'application. La première est basée sur un ordonnancement à priori des transactions par "time-stamps" (BERN80). La seconde solution, plus lourde, consiste à vérouiller successivement les pages (ESWA76). Finalement, les pannes sont contrôlées par deux moyens : duplications de matériel et logiciel et atomicité des transactions. D'une part, au moins deux processeurs sont assignés à une fonction et, en cas de panne de l'un, la fonction

est toujours disponible et la machine bases de données fonctionne en mode dégradé. D'autre part, une transaction est atomique, c'est à dire que ses mises à jour seront toutes ou nullement prises en compte selon que la transaction ait été correctement terminée ou non.

2-7. Sécurité des données

SABRE permet de protéger les données contre les accès non autorisés. Afin de contrôler ces accès, l'administrateur de la base de données doit pouvoir spécifier qui peut effectuer quelles opérations sur quelles données de sa base. Deux mécanismes classiques réalisent ces protections (DEMI78) : la possibilité de définir des vues de la base et le contrôle des droits d'opérations. Les vues définissent des sous-ensembles de la base de données et sont attribués par l'administrateur. Les droits d'opérations sur les données sont accordés à des transactions qui devront être authentifiées lors du contrôle des droits afin d'être autorisées. Une transaction peut transmettre ses droits à d'autres transactions (GRIF76).

3. PRINCIPES DE BASE

3-1. Partitions associatives

Les données sont stockées dans des partitions associatives. Une partition est définie comme un granule de mémoire secondaire adressable par contenu. Les données sont insérées séquentiellement et retrouvées en spécifiant un critère multi-attribut, qui précise une information sur le contenu des attributs et non leurs adresses. Compte tenu de la technologie actuelle, la partition est assimilée à une piste de disque. A l'intérieur de la partition, la notion d'adresse est inexistante, ce qui permet les mouvements de données dans une partition sans aucune mise à jour des tables contenant les chemins d'accès (index, arbres, liens, ...). L'unité de la partition adressable par associativité est l'attribut. L'occurence d'attribut (une valeur particulière de l'attribut) est une donnée de taille variable. L'unité d'insertion dans une partition est le tuple, qui est un groupement d'occurences de différents attributs. Les tuples sont donc aussi de taille variable. L'insertion d'un tuple dans une partition peut provoquer l'éclatement de cette partition en deux, si celle-ci est pleine.

3-2. Filtrage

Le filtrage est la fonction qui réalise l'associativité. Lorsqu'une piste est lue sur le disque, le flot de données est simultanément filtré par un processeur, appelé filtre, associé a l'unité de disque (FAUD80). Ainsi, seuls les attributs répondant au critère de recherche spécifié sont sélectionnés. Le filtrage est utilisé pour les opérations relationnelles unaires de restrictions et projections (sans élimination des doubles).

3-3. Regroupement des données

Les données de la base sont placées de facon a limiter l'espace de recherche et réduire l'effort nécéssaire pour retrouver les données qui doivent satisfaire un critère donné. La base de données est partitionnée sur un ensemble d'unités de disques accessibles en parallèle par les filtres qui leur sont associés. La stratégie de placement de SABRE doit assurer que les tuples répondant à une question donnée puissent être retrouvés dans un nombre minimal de partitions accessibles en parallèle. Pour ce faire, une technique de placement par regroupement multi-attribut est proposée dans SABRE (KARL81). Pour chaque relation a placer, un ensemble ordonné d'attributs (A1, A2, ..., An) est donné par l'administrateur. Pour chaque attribut Ai, l'alphabet dans lequel l'attribut Ai prend ses valeurs est défini de différentes facons possibles (énumération des valeurs, intervalle, hachage ...). A chaque attribut Ai correspond un niveau de regroupement i, le premier niveau étant celui de l'attribut A1. Pour un niveau i, les tuples avant des valeurs voisines d'attribut Ai sont regroupés dans des partitions parallèles. Si la relation est partitionnée sur d disques, ceci assure que les tuples répondant à un critère de recherche spécifiant Ai seront retrouvés en parallèle sur les d disques, divisant ainsi le temps d'exécution par d. Lorqu'une partition ne contient plus que des tuples de même valeur d'attribut Ai, la partition est éclatée et l'attribut Ai+1 est utilisé pour le regroupement suivant. Finalement, les éclatements successifs de partitions à différents niveaux suivant différents attributs conduisent à la construction d'un arbre de recherche multi-dimensionnel. La structure de l'arbre sera implantée sous forme d'arbre couvrant réduit, ayant pour principale propriété un très faible coût de stockage et d'accès (KARL81). Les arbres couvrants réduits serviront aussi à la réalisation d'index secondaires.

3-4. Opérations parallèles

Certaines opérations de base des système de gestion de base de données relationnels restent tres coûteuses en temps d'execution. Ce sont les opérations binaires, qui nécéssitent un traitement répétitif sur de grandes relations qui ne peuvent être contenues en mémoire. Généralement, chaque étape de traitement exige la lecture et (ou) écriture de la relation. Ces opérations sont : tri externe, jointure, élimination des doubles. Cette dernière opération suit l'opération de projection telle que la réalisent les filtres. Des algorithmes multiprocesseurs pour réaliser ces opérations (VALD80, VALD81b, VALD81c) seront présentés aux chapitres III, IV et V et constituent la principale contribution à la recherche de cette thèse. L'objectif est d'atteindre pour ces opérations des coûts inversement proportionnels au nombre de processeurs traitants, en développant des méthodes qui profitent au mieux du parallelisme inhérent a SABRE.

3-5. Décomposition de questions

Une requête de l'utilisateur est exprimée en un langage assertionnel de type QUEL (STON76). Elle est décomposée en opérations de base de l'algèbre relationnelle générant ainsi un arbre ou les noeuds représentent les opérations et les branches l'enchainement des opérations. La question ainsi décomposée est optimisée afin de diminuer son coût global et est traduite en un arbre ou les opérations sont effectuées par des opérateurs SABRE. Par exemple, l'opération de restriction-projection devient une opération de sélection, réalisée par les filtres, qui peut être suivie d'une opération d'élimination des doubles. La fonction d'optimisation de questions doit maximiser le parallèlisme et minimiser le temps d'entrées-sorties. Un premier algorithme de décomposition et d'optimisation de question est proposé dans (BELL80).

3-6. Mises à jour différées

SABRE n'autorise pas la mise à jour directe de la base. La définition d'une relation temporaire est nécessaire pour contenir les données mises à jour. Les relations temporaires mises à jour sont intégrées aux relations permanentes à la fin de transaction, autorisant ainsi la mise à jour de la base. En conséquence, les mises à jour sont différées à la fin de la transaction, ou elles sont effectuées sous contrôle d'un protocole de "commit" en deux étapes (BAER80).

3-7. Protocole de manipulation de données

Finalement, SABRE présente une interface externe qui peut être considérée comme un protocole situé au niveau application de l'architecture ISO des systèmes ouverts (ISO79). Ce protocole précise toutes les interactions entre SABRE et les calculateurs hôtes. Une première version de ce protocole est présentée dans (GARD80b).

4. ARCHITECTURE DE SABRE

4-1. Architecture fonctionnelle

L'architecture fonctionnelle décrit SABRE en termes de niveaux et d'interfaces entre ces niveaux. Cette architecture, composée de cinq niveaux, est représentée figure 1. Chaque niveau manipule des objets de plus en plus fins. Ces niveaux sont :

(1) Base de données répartie (BDR) :

Ce niveau est optionnel. Il permet de réaliser les fonctions d'intégration de SABRE dans un système de gestion de base de données répartie. Ce niveau assure la traduction des requêtes portant sur des données réparties en requêtes portant sur des vues locales.

(2) Vue :

Le niveau gestion de vues permet la définition et manipulation de schémas externes, afin d'assurer la sécurité. Il traduit une requête portant sur des vues externes en une ou plusieurs requêtes manipulant les relations implantées. De plus, il réalise le contrôle des droits d'accès des transactions.

(3) Bases de données locales :

Ce niveau de base fournit les outils nécéssaires à la gestion du schéma des bases de données locales et exécute la décomposition de question et son optimisation.

(4) Relations :

Ce niveau gère les chemins d'accès aux données et traduit un nom de relation associé à un prédicat en un ensemble d'adresses de partitions où se trouvent des tuples répondant au critère. Ce niveau gère le placement des tuples par regroupement multi-attribut.

(5) Partitions :

Ce niveau réalise les opérations parallèles sur l'ensemble des partitions fourni par le niveau 4. Ces opérations sont sélection, tri, jointure, et élimination des doubles. Ce niveau inclut aussi la gestion de la mémoire cache.

Un niveau est décrit par son interface externe et par l'interface externe du niveau supérieur. L'interface externe du niveau 1 est le langage de manipulation de données. Les interfaces externes des niveaux 2 et 3 sont une forme compilée déduite du langage de manipulation de données. L'interface externe du niveau 4 est l'algèbre relationnelle. L'interface externe du niveau 5 est aussi l'algèbre relationnelle au

niveau d'ensembles de partitions.

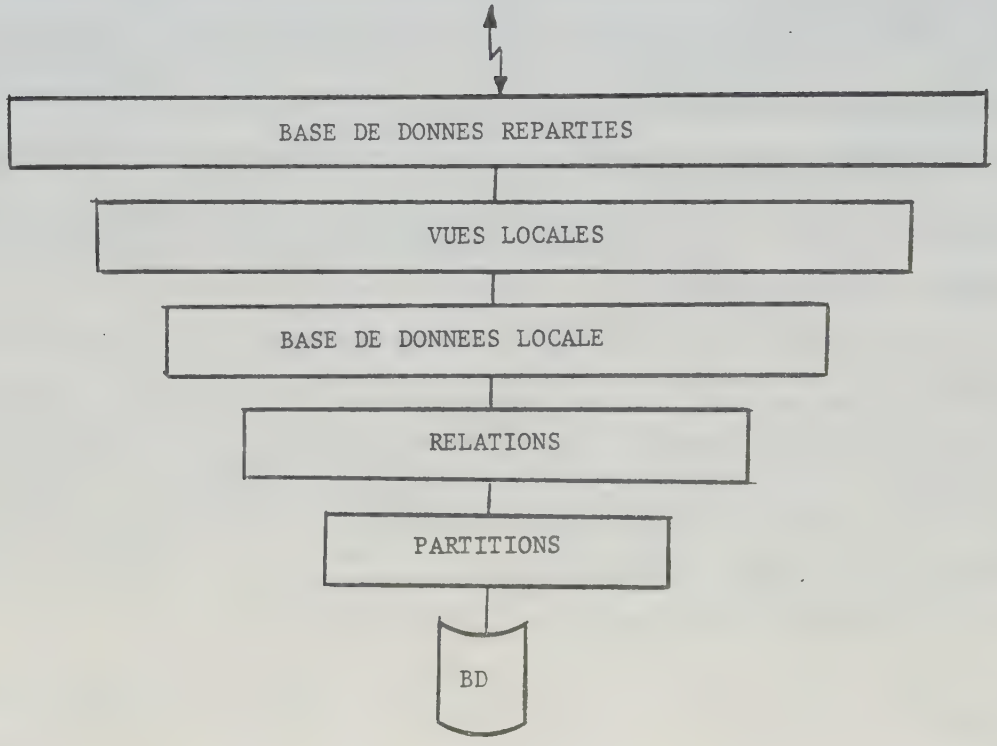


figure 1 : Architecture fonctionnelle

4-2. Architecture opérationnelle

L'architecture opérationnelle est composée d'un ensemble de processeurs communiquant au moyen d'un réseau d'interconnexion. Nous décrivons maintenant brièvement chacun des processeurs qui seront inclus dans SABRE :

(1) Station de transport :

Le processeur station de transport (PST) réalise les fonctions de base du niveau BDR, qui sont l'émission et la réception de messages. C'est le seul processeur actuellement spécifié à ce niveau. Il est prévu un autre processeur optionnel offrant l'intégration de SABRE dans un système de gestion de base de données répartie.

(2) Processeur de sécurité et vues :

Ce processeur (PSV) correspond à l'implantation du niveau vues locales.

(3) Décomposeur :

Ce processeur de manipulation de base (PMB) réalise la décomposition et l'optimisation de question. Il correspond à l'implantation du niveau base de donnée locale.

(4) Accélérateur :

Ce processeur de manipulation de relations (PMR) gère les chemins d'accès aux relations et correspond à l'implantation des fonctions associées au niveau relation.

(5) Processeur de fiabilité et reprise (PFR) :

Ce processeur doit maintenir la fiabilité des données. Il assure la gestion de journaux où sont enregistrées les partitions mises à jour, recoit les points de reprise et exécute les reprises après pannes.

(6) Gérant de mémoire cache :

Le processeur gérant la mémoire cache (PMC) a pour rôle la gestion des relations intermédiaires et résultats. Il anticipe les lectures de partitions sur disques et alloue des pages (unités de mémoire cache) aux autres processeurs pour stocker des résultats (après filtrage, jointure etc). De plus, la mémoire cache est étendue sur mémoire secondaire lorsque des pages libres sont nécéssaires. Ce processeur représente une partie de l'implantation du niveau partition.

(7) Processeur de jointure et tri (PJT) :

Ces processeurs réalisent les opérations parallèles de jointure, tri ainsi qu' élimination des doubles et semi-jointure. Les algorithmes multiprocesseurs décrits dans (VAL80, VALD81b, VALD81c) seront implantés dans chaque PJT. Les PJTs représentent une autre partie de l'implantation du niveau partition.

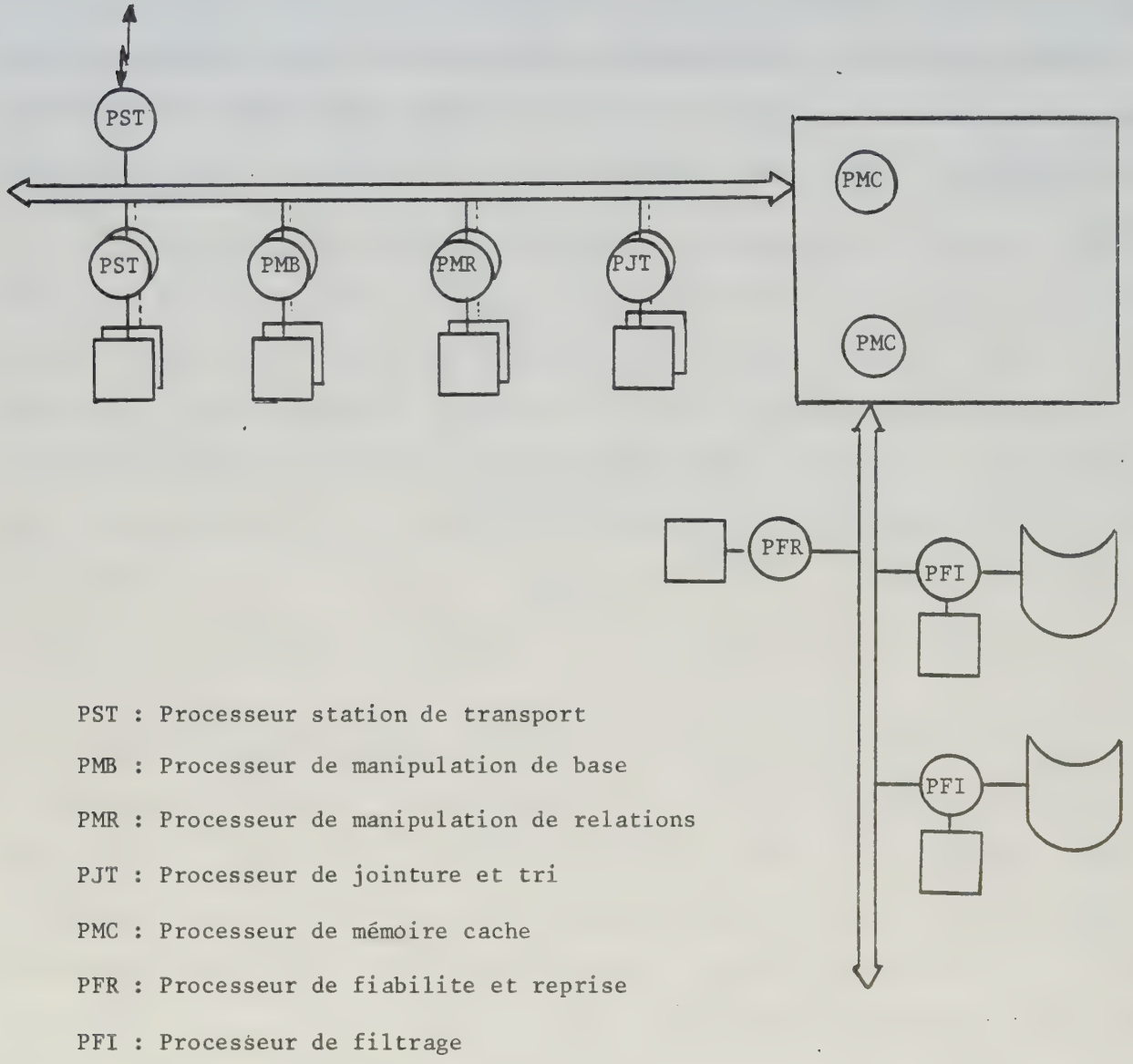
(8) Filtre :

Un processeur de filtrage (PFI) sera associé à chaque unité de disque, c'est à dire à chaque flot de données provenant d'un disque. Pour ce faire, une nouvelle classe de filtres dérivés de la forme générale des automates à états finis est proposée dans (FAUD80). Les filtres permettent l'exécution de différentes questions simultanément. Ils représentent la dernière partie de l'implantation du niveau partition.

4-3. Réseau d'interconnexion

Les différents processeurs de l'architecture opérationnelle sont reliés par un réseau d'interconnexion. Différents procédés peuvent être envisagés: multi-bus (INTE79), multi-boucles (BANE78), matrice (DEWI79). Les critères de choix retenus sont : éviter les goulots d'étranglement dus aux communications entre processeurs , et être faisable avec la technologie actuelle. Des évaluations et simulations seront faites pour qualifier le réel besoin de réseau d'interconnexion dans SABRE. Le réseau d'interconnexion désiré comporterait deux bus reliés à la mémoire cache et un dispositif d'interconnexion spécifique pour les processeurs de jointure et tri. Le premier bus relierait un ensemble de filtres associés aux unités de disques avec la mémoire cache pour constituer une machine physique. Cette machine physique est reliée à un ensemble de processeurs (PFR, PMR, PMB, PSV, PST) par le second bus. Le réseau d'interconnexion entre les processeurs de jointure et

tri, et la mémoire cache doit avoir deux propriétés. La première est que différents PJT puissent lire ou écrire des pages en mémoire cache simultanément. La seconde est de pouvoir diffuser la même page à un nombre quelconque de PJT en un seul transfert. Ce type d'interconnexion est faisable (DEWI79), et différentes alternatives sont envisagées. Le développement actuel de SABRE s'efforce d'être indépendant du réseau d'interconnexion. Une vue générale de l'architecture est présentée figure 2.



PST : Processeur station de transport

PMB : Processeur de manipulation de base

PMR : Processeur de manipulation de relations

PJT : Processeur de jointure et tri

PMC : Processeur de mémoire cache

PFR : Processeur de fiabilite et reprise

PFI : Processeur de filtrage

figure 2 : L'architecture opérationnelle de SABRE

4-4. Le noyau d'exécution réparti

Le noyau d'exécution réparti (NER) a pour rôle l'allocation et le contrôle des processeurs. C'est un système opératoire distribué sur chacun des processeurs permettant le contrôle de l'exécution répartie des questions. L'exécution d'une question d'un utilisateur dans SABRE procède essentiellement ainsi. Après réception par la station de transport (PST), la question exprimée en langage assertionnel est contrôlée et modifiée afin de permettre les conversions de données (STON76) par le processeur de sécurité et vues (PSV), puis elle est transmise au décomposeur (PMB). La question est alors traduite en opérations de l'algèbre relationnelle et le décomposeur produit un plan d'exécution composé d'une suite d'actions à effectuer (sélections, jointures ...) et d'expressions de synchronisation. Le NER évalue les expressions de synchronisation et initialise les actions dont les conditions de synchronisation sont vérifiées. Chaque action ainsi initialisée est exécutée par un ou plusieurs processeurs assignés a cette fonction.

Considérons l'exemple d'une question invoquant deux sélections dans des relations permanentes puis la jointure des tuples sélectionnés. Le plan d'exécution précisera deux actions de sélection et une action de jointure conditionnée par une expression de synchronisation. Les actions de sélection sont transmises à l'accélérateur (PMR) qui convertit le nom de relation associé à un prédicat de sélection en une suite d'adresses de partitions. Les actions de sélection sont alors déclenchées par les filtres, générant ainsi deux relations temporaires en mémoire cache. La dernière action du plan est la jointure des relations temporaires. Elle est initialisée lorsque l'expression de synchronisation indiquant que les premiers résultats de sélection sont disponibles en mémoire cache est vérifiée. Après cette dernière action,

l'utilisateur pourra demander lecture de ses résultats. Lorsqu'une action est terminée, le processeur qui représente la fonction retourne au NER les paramètres résultats et la réponse, afin que de nouvelles actions puissent être activées. Chaque fonction est représentée par un processeur, dont la couche NER maintient une file d'attente des actions a effectuer. Ainsi l'exécution répartie d'une requête d'un utilisateur peut être contrôlée. Pour plus de détails, le NER est présenté dans (BERN81).

5. CONCLUSION

Dans ce chapitre, le contexte de cette thèse, la machine bases de données SABRE a été présenté. Les objectifs, les principes de base et , finalement, l'architecture de SABRE ont été décrits. SABRE est une machine bases de données multiprocesseur qui s'efforce d'intégrer toutes les fonctions de gestion de grandes bases de données relationnelles. Les objectifs sont des gains essentiels en performances et fonctionnalité, sans pour cela révèler le besoin d'un matériel particulier et coûteux. Le projet explore de nouveaux concepts tels que parallélisme, filtrage, regroupement multi-dimensionnel des données et tente de maitriser les fonctions d'intégrité et de sécurité des données. Le concept de parallélisme et d'algorithmes multiprocesseurs sera développé en détail dans les chapitres suivants.

La distinction entre architecture fonctionnelle et architecture opérationnelle permet l'indépendance entre fonctions a réaliser et logiciel pour les réaliser. L'architecture fonctionnelle est composée de niveaux manipulant des granules de plus en plus fins, et d'interfaces entre ces niveaux. A chaque niveau de l'architecture fonctionnelle correspond un ou plusieurs processeurs de l'architecture opérationnelle. Les processeurs sont contrôlés par un noyau d'exécution réparti en une

couche de logiciel sur chacun des processeurs. Un dernier objectif du projet SABRE est de proposer des solutions originales pour les problèmes posés (BELL80, BERN81, FAUD80, KARL80, KARL81, KUBR81, TEMM81, VALD80, VALD81a, VALD81b, VALD81c, VIEM81).

Une première version monoprocesseur de SABRE est opérationnelle à l'INRIA, programmée en PASCAL sous le système MULTICS. Elle intègre quatre processeurs (fonctionnels) : un processeur simulant l'ordinateur hôte (interface QUEL), un processeur de décomposition de question, un processeur de jointure et tri, et un processeur simulant la mémoire cache. Cette première version a été démontrée lors des journées de présentation des résultats du projet SIRIUS les 26 et 27 novembre 1981.

CHAPITRE II :

ANALYSE DE DIVERSES METHODES DE JOINTURE DE RELATIONS

1. INTRODUCTION

Plusieurs systèmes basés sur le modèle relationnel (CODD70) sont opérationnels (BABB79, CHAM81, STON76) ,et posent toujours les problèmes d'optimisation de coût de stockage, d'accès, et plus généralement de performances des manipulations de relations. Ces coûts sont d'un grand intérêt lors de l'évaluation des systèmes et sont finalement à prendre en compte pour comparer le modèle relationnel avec d'autres modèles de données. Des méthodes optimisées évaluant les questions par décomposition en opérateurs de l'algèbre relationnelle ont été proposées (YAO79,SMIT75,PECH75). Les opérations de base de l'algèbre relationnelle résultant de cette décomposition sont restrictions, projections, et jointures (DATE75). C'est en particulier la réalisation de ces opérateurs qu'il faut optimiser.

Une opèration de l'algèbre relationnelle très coûteuse en temps d'execution est la jointure. Elle nécéssite l'accès à deux relations de la base de données choisies par un algorithme de décomposition et d'optimisation de question (AHO79). Joindre deux relations équivaut à faire l'intersection de deux ensembles, où chaque élément du premier ensemble doit être comparé à chaque élément du deuxième.Ceci peut conduire pour des ensembles de n eléments à un coût de n**2 comparaisons. Ce nombre peut être réduit jusqu'à n *log n en ordonnant un ou les deux ensembles. D'autres méthodes consistent à utiliser des chemins d'accès particuliers (index secondaires, fichiers aléatoires, fichiers inversés, etc) ou (et) à tirer profit de la nature des attributs joints (par exemple lorsqu'ils sont clés).

Ce chapitre présente une synthèse et une comparaison des principales méthodes proposées pour traiter éfficacement l'opération relationnelle de jointure. La synthèse des méthodes existantes présente l'état de l'art. Des algorithmes implantès dans des systèmes existants

ou seulement décrits sont étudiés et évalués en tenant compte du coût d'entrées-sorties et du coût de stockage. Outre l'aspect synthèse, l'originalité essentielle de ce chapitre est d'apporter une comparaison rigoureuse des différents algorithmes en fonction de divers paramètres : taille des relations,sélectivité des attributs , présence d'index,attributs de jointure clés.

Les paramètres nécéssaires aux évaluations sont d'abord présentés avec des hypothèses qui permettront de simplifier quelque peu les formules. Les algorithmes implantés dans les systèmes INGRES, R , C.A.F.S. et ceux proposés dans (GOTL75) utilisent différentes stratégies pour réaliser la jointure. Ces méthodes emploient des tris externes dont l'intéret a été souligné dans (DEMO80) ,des fonctions de hachage ou des index temporaires. Ces algorithmes sont tout d'abord présentés et évalués dans les trois cas suivants:

- les attributs de jointure ne sont pas des clés et il n'existe pas d'index secondaire,
- il existe des index secondaires sur les attributs de jointure,
- les attributs de jointure sont des clés. L'évaluation se base sur le nombre d'accès aux mémoires secondaires, considéré comme le paramètre de performance le plus critique dans le traitement de la jointure.

Ensuite,les différentes méthodes examinées sont comparées et il est montré que l'éfficacité d'une méthode depend essentiellement de la valeur des paramètres.

2. NOTATIONS.

Nous considèrons la jointure des relations R et S comportant respectivement m et n pages sur les attributs A et B. Le prédicat de jointure applicable aux attributs A et B est noté θ.

Les unités physiques sont des pages qui peuvent contenir des tuples ou des informations sur les index. Les pages contiennent les tuples d'une même relation. Un identifieur (TID) est associé à chaque tuple permettant de retrouver la page de ce tuple en un accès. La distribution de ces tuples est supposée uniforme dans l'espace disque. Nous supposerons aussi la répartition uniforme des valeurs distinctes d'attributs de jointures. Afin de comparer les algorithmes dans les mêmes conditions, le coût pris en compte pour l'évaluation des algorithmes est celui de la jointure, sans les opérations de restrictions et projections. Le coût d'écriture de la relation résultat est supposé le même pour tous ces algorithmes et sera omis lors de l'évaluation.

Les index utilisés peuvent être décrits sous plusieurs formes. La forme la plus simple est un ensemble de couples (valeur d'attribut, TID) représentée par un B-tree. La deuxième forme est un ensemble de couples (valeur d'attribut, liste de TIDs), ces TIDs pointant sur les tuples ayant cette valeur d'attribut. Ceci évite de répéter l'information valeur d'attribut pour les synonymes. La dernière forme est représentée par des index hachés ou ISAM.

Les autres paramètres nécessaires aux évaluations sont les suivants

El(respectivement E2) : le nombre moyen de tuples de R(respectivement S) contenus dans une page,

nj(resp.mj) : la proportion de tuples de R(resp.S) qui participent a la jointure,

K : le nombre de tuples par page, supposé égal pour les 2 relations (K=E1=E2),

I : le nombre de pages d'un index.

M : le nombre de valeurs différentes d'attributs de jointure pour une relation (= nombre d'éléments de la forme (VALEUR,TETE DE LISTE D'ADRESSES) d'un index secondaire).

N : la taille mémoire allouée à la jointure (en nombre de pages).

3. CAS SANS CHEMINS D'ACCES.

3-1. Méthode des boucles imbriquées.

La méthode la plus triviale (GOTL75) consiste à composer le produit cartésien des deux relations,où chaque élément de la première est comparé à chaque élément de la deuxième. Cet algorithme est représenté figure 1.

```
procedure JOINDRE1 (R,S);

   pour chaque  tuple de R
     faire
       pour chaque  tuple de S
         faire
           si r.AΘs.B alors former la jointure.
fin-JOINDRE1
```

figure 1 : algorithme par produit cartésien .

Une telle procédure exige m+mn accès sur mémoire secondaire,ce qui est en général prohibitif. D'où l'intérêt à développer des méthodes plus optimisées.

3-2. Méthode d'INGRES (STON76).

INGRES ne possède pas d'opérateur de jointure clairement défini. Celui-ci est confondu avec l'algorithme de décomposition de question. Cependant,dans le cas étudié, il est possible d'en déduire une méthode

pour effectuer la jointure. Cette méthode consiste à créer un fichier aléatoire pour la plus petite relation. Ce fichier contient les tuples projetés sur les attributs de jointure et les attributs désirés dans le résultat et est haché sur la clé obtenue par concaténation des attributs de jointure. Chaque valeur d'attribut de jointure est alors insérée dans une liste ordonnée W. La deuxième relation est ensuite balayée séquentiellement. Pour chaque tuple, si la valeur d'attribut de jointure est présente dans W, un accès direct au fichier aléatoire permet de composer la jointure, en supposant une distribution uniforme des valeurs d'attributs de jointure,qui évite les débordements. L'algorithme est décrit plus en détail figure 2.

```
algorithme : (avec m n)

1) BALAYER S
    pour chaque tuple s de S
        INSERER (s.B,W)
        ECRIRE (s,h(s.B))
2) BALAYER R
    pour chaque tuple r de R
      si "r.A est dans W"
        LIRE (h(r.A))
        si "acces valide"
            pour chaque tuple du paquet
               si r.A θ s.B alors JOIN (r,s)
```

figure 2 : algorithme d'INGRES.

Figure 2 , ECRIRE (s,h(s.B)) est une procèdure qui écrit le tuple s dans un paquet du fichier haché,défini par l'application de la fonction h appliquée à l'attribut de jointure B. INSERER (B,W) permet d'insérer en ordre la valeur B dans la zone W. LIRE (h(r.A)) est une procèdure qui lit le paquet d'adresse h(r.A). JOIN (r,s) est une procèdure qui écrit un tuple de la relation résultat ,en concaténant les tuples r et s.

La création du fichier haché nécèssite l'utilisation de toute la place mémoire allouée à la jointure de sorte que tous les paquets se trouvent en partie en mémoire. Si un paquet est de la taille d'une

page,l'écriture de ce paquet éxige n/N accès MS et l'écriture de tout le fichier : n*n/N accès MS. Avec la lecture de la relation S , l'étape 1 coûte : n+(n*n)/N acces MS. La 2eme étape,jointure proprement dite,est ainsi très performante car elle nécèssite un balayage de la relation R (m accès) et un accès direct aux tuples contenus dans le fichier haché,qui appartiennent à la jointure (mj accès). Cette étape éxige donc : m+mj accès MS. Le coût total de cet algorithme est alors:

$$t(2) = n + n*n/N + m + mj .$$

Il est à noter que si N, paramètre de la taille mémoire est suffisamment grand ,cette méthode est très éfficace et a un coût de l'ordre de m+n entrées-sorties.

3-3. Méthodes de SYSTEM-R (ASTR76)

La méthode d'optimisation du System-R a été présentée dans (SELI79). Pour réaliser la jointure ,SYSTEM-R a le choix entre un ensemble de méthodes (BLAS77). Ce choix est réalisé par l'optimiseur qui tient compte des possibilités d'accès aux relations,de leur groupement en mémoire secondaire, et de l'éxistence de liens entre relations. Dans le cas étudié ,deux méthodes sont possibles :

- tri des deux relations selon l'attribut de jointure,
- méthode multipasse.

3-3-1. Tri des relations

Cette méthode emploie le tri des deux relations sur l'attribut de jointure suivi de leur fusion sur l'attribut de jointure pour réaliser l'opération. Les relations à joindre sont parcourues pour créer deux fichiers de travail ne contenant que les sous-tuples répondant aux restrictions. Les fichiers temporaires ainsi créés sont triés selon l'attribut de jointure par une méthode de tri-fusion à deux voies. Il suffit ensuite de parcourir ces fichiers pour constituer la relation

résultat par une opération de type fusion. Cette méthode est décrite par la figure 3.

algorithme par tri

1) BALAYER les 2 relations (en éffectuant les restrictions).
2) CREER 2 fichiers contenant la projection des tuples sur les attributs de jointure et résultats.
3) TRIER ces 2 fichiers sur les attributs de jointure.
4) BALAYER simultanément en fusionnant ces 2 fichiers et former la jointure.

figure 3 : méthode par tri.

Le coût de la jointure,isolé des opérations de restrictions et projections, comporte le coût des tris (étape 3) et le coût de la fusion (étape 4). Trier un fichier de n pages requiert un nombre de l'ordre de log n passes (KNUT72),(tous les log sont à base 2,pour un tri-fusion d'ordre 2). Le nombre d'accès MS pour trier est donc 2n*log n car il faut une lecture et une écriture de tout le fichier à chaque passe.

La fusion consistant en une lecture de chaque fichier coûte m+n accès MS. Le coût total de cette méthode est finalement:

$$t(3) = 2m*\log m + 2n*\log n + m + n.$$

Il est à remarquer que l'algorithme de tri s'optimise en utilisant une grande zone de travail. L'utilisation d'un bloc de O pages comme unité de transfert réduit ce coût a 2n/O*logn/O accès MS.

3-3-2. Méthode multipasse.

Cette méthode utilise un parcours multiple des deux relations. La première relation est répartie dans une zone de travail W, de sorte que les tuples correspondants aux plus petites valeurs d'attribut de jointure y soient insérés. La deuxième relation est balayée en comparant chaque tuple à ceux de W pour effectuer la jointure. Puis W est à nouveau chargé pour les tuples écartés lors de la première passe

et le traitement est répété.

Une description plus précise de l'algorithme est présentée ci-dessous. MAX contient toujours la plus grande valeur de l'attribut de jointure pour les tuples déja insérés dans W. INSERER est une procédure permettant d'insérer un tuple dans W alors que ECARTER permet de rejeter un tuple et de l'enlever de W s'il y est déja.

```
algorithme multipasse

1) BALAYER S en effectuant d'éventuelles restrictions.
   pour chaque tuple s de S
      - PROJETER sur les attributs de jointure et résultats
      - si "place dans W" alors  INSERER (s)
        sinon
          si s.A < MAX
           alors
              - s'="dernier tuple dont l'attribut de jointure
                a la valeur MAX"
              - ECARTER (s')
              - INSERER (s)
          sinon  ECARTER (s).

2) BALAYER R en effectuant d'éventuelles restrictions.
     CHERCHER dans W les s qui se joignent.

3) Répéter le traitement pour les tuples écartés vérifiant
      "attribut de jointure>MAX"
```

figure 4 : algorithme multipasse.

Il faut m+n accès MS pour chaque passe,ce qui fait,si p est le nombre de passes : p(m+n) accès MS Cet algorithme est très efficace et est choisi par l'optimiseur si une des relations nécessaire à la jointure tient en mémoire W. Il demeure intéressant tant qu'il ne nécessite pas de répéter un nombre important de passes (par exemple pour p<3).

3-4. Méthode de C.A.F.S. (BABB79).

CAFS est une machine bases de données disposant d'un nouvel hardware permettant l'exécution rapide de questions invoquant des jointures. CAFS utilise un tableau de bits pour retrouver rapidement les

données. Deux algorithmes de jointure sont décrits. Le premier utilise le codage des valeurs d'attributs et le second des fonctions de hachage. Ces algorithmes ne permettent que d'obtenir des résultats portant sur l'une des relations,c'est à dire d'éffectuer des semi-jointures (PALE72,BERN79a). Nous proposons ensuite une méthode simple pour réaliser la jointure à partir de deux semi-jointures ,pour pouvoir comparer cette méthode avec d'autres.

3-4-1. Semi-jointure avec attributs codés.

Chaque valeur d'attribut de jointure est codée de 1 a N. Un attribut est stocké sous forme codée. Un tableau de bits M(1:N) est associé à l'algorithme. Cet algorithme est décrit par la procèdure SEMI-JOINDRE1 figure 5.

```
procedure SEMI-JOINDRE1

1) EFFACER M
2) BALAYER R
    pour chaque tuple (répondant a la restriction)
     faire
       I= valeur de l'attribut de jointure
       M(I) = 1
3) BALAYER S
    pour chaque tuple
     faire
       I=valeur de l'attribut de jointure
       si M(I)=1 alors conserver les résultats de S
fin.
```

figure 5 : algorithme de semi-jointure 1.

Exemple

Soit la base de données composée de deux relations EMPLOYE (nom, salaire, service) et SERVICE (service, nom chef, salaire), l'application de l'algorithme à la semi-jointure de ces deux relations sur l'attribut service est représentée ci-dessous. L'attribut IS, code de l'attribut service, est ajouté à chacune des relations lors de la compilation de la question, la correspondance se faisant au moyen d'un dictionnaire. Ceci est décrit figure 6.

EMPLOYE

nom	salaire	service	IS
DUPONT	5000	VENTE	1
DURAND	4000	ACHAT	2
DUPUY	3000	VENTE	1

SERVICE

IS	service	chef	salaire
2	ACHAT	PIERRE	6000
1	VENTE	JEAN	6200

figure 6 : exemple de semi-jointure utilisant le codage d'attribut

Soit la question : quel est le chef de Durand ?

L'application de l'algorithme est la suivante :

étape 2 : lecture de la relation EMPLOYE

M(2) est mis a 1

étape 3 : lecture de la relation SERVICE

le test M(2) = 1 est vérifié pour PIERRE

Evaluation de l'algorithme

Cette procédure nécessite seulement le balayage de chaque relation, ce qui coûte m+n entrées-sorties.

Cette méthode nécessite le stockage du tableau M et pour tout attribut d'une relation,le stockage sous forme codée des attributs,qui peut être un facteur important d'overhead si la proportion des valeurs différentes d'attributs est grande. L'utilisation de cette procédure n'a d'intérêt qu'à partir d'un certain nombre de valeurs d'attributs synonymes.

3-4-2. Jointure avec des fonctions de hachage

Dans ce cas,la valeur d'attribut n'est plus codée. La correspondance est faite au moyen de fonctions de hachage appliquées sur la valeur K des attributs de jointure. Un tableau de bits M(1:N) est utilisé et M(h(K)) est positionné à 1 lors de la lecture de la première relation. Pour réduire les risques de collisions avec des valeurs d'attributs de jointure différentes K1 et K2 telles que h(K1)=h(K2), p

fonctions de hachage associées à p tableaux de bits peuvent être utilisées. L'algorithme est décrit par la procèdure SEMI-JOINDRE2 figure 7.

```
procedure SEMIJOINDRE 2

1) EFFACER M1 ,M2,...Mp
2) BALAYER R
   pour chaque tuple répondant ala restriction
     faire
        - M1(h1(K))=1,M2(h2(K))=1,...Mp(hp(K))=1
        - mettre les attributs de jointure dans une liste P'
3) BALAYER S
   pour chaque tuple répondant a la restriction
    et si M1(h1(K'))=1 et M2(h2(K'))=1 et...Mp(hp(K'))=1
        - mettre les attributs de jointure et les attributs
          projetés dans la liste P
4) ELIMINER dans P les attributs qui ne sont pas dans P'
```

figure 7 : algorithme de semi-jointure 2.

Cet algorithme est le même que le précédent à la différence de la 2ème ligne ajoutée au step 2 et l'ajout du 4ème step,qui ont pour but d'éliminer les collisions. Là le bilan en accès MS est le même qu'en 3-4-1: m+n. Mais, le coût de stockage supplémentaire de l'algorithme précédent a disparu.

Jointure avec deux semijointures:

Nous proposons une méthode simple pour réaliser la jointure avec cette procédure ,qui est suffisamment performante pour permettre la comparaison avec d'autres méthodes mais n'est cependant pas optimale.

1) composer les semijointures de R et S dans deux fichiers P1 et P2.
2) trier P1 et P2 sur l'attribut de jointure.
3) fusionner P1 et P2 dans la relation résultat.

Le coût de l'étape 1 est 2m + 2n. Le coût de l'étape 2 est 2 nj/K *log(nj/K) + 2 mj/K *log(mj/K) car ni et mj sous-tuples de P1 et P2 sont triés avec un tampon de taille K. Le coût de l'étape 3 est ni/K

+ mj/K pour lire chaque semi-jointure. Le coût total des accès MS est enfin:

t(SJ) = 2m +2n +2nj/K*log(nj/K) +2mj/K*log(mj/K) +nj/K +mi/K.

Le facteur dominant est 2m+2n,le reste représentant pour une jointure moyenne avec un coefficient de sélectivité de jointure de 0.2 environ 10% de ce facteur.

4. CAS AVEC INDEX SECONDAIRES

4-1. Algorithme de Gotlieb

Leo R.Gotlieb propose l'algorithme suivant lorsqu'il éxiste un fichier inversé sur l'attribut de jointure pour une des deux relations. L'idée est de construire dynamiquement un fichier inversé sur l'autre relation, ordonné comme le premier. Pour simplifier,nous traitons l'équijointure.

Soit le fichier inversé sur S,INV-S défini ainsi

```
01  INV-S (1:M)
    02  BVAL  : valeur d'attribut
    02  BLIST : adresse de la liste contenant des adresses
        de tuples de S tels que "s.B = BVAL"
```

M est le nombre d'attributs de jointure de valeurs différentes pour la relation S,et pour chacune de ces valeurs,la liste rangée à l'adresse BLIST contient les adresses des tuples ayant la valeur d'attribut de jointure BVAL. Un fichier inversé sur la 2ème relation est construit : MATCH (1:M). MATCH est ordonné comme INV-S et rempli pour les valeurs d'attribut se joignant et donnant pour MATCH (i) la tête de liste des adresses de tuples de la 2ème relation ayant INV-S(i).BVAL pour valeur d'attribut. L'organisation des fichiers inversés est shématisée figure 8. L'algorithme comporte deux parties: la construction de MATCH puis le balayage simultané des fichiers inversés pour faire la jointure. Pendant la construction du fichier MATCH, M tampons sont utilisés (un

par BVAL différente) et chacun pouvant contenir J adresses de tuples. La relation résultat décrite dans (GOTL75) est constituée par des couples de pointeurs vers les tuples de chaque relation qui participent à la jointure. Cependant, dans l'évaluation de l'algorithme,nous ajouterons le cout d'accès à ces tuples par les pointeurs afin de composer une relation résultant de la concaténation de ces tuples. Cet algorithme est plus précisément décrit figure 9.

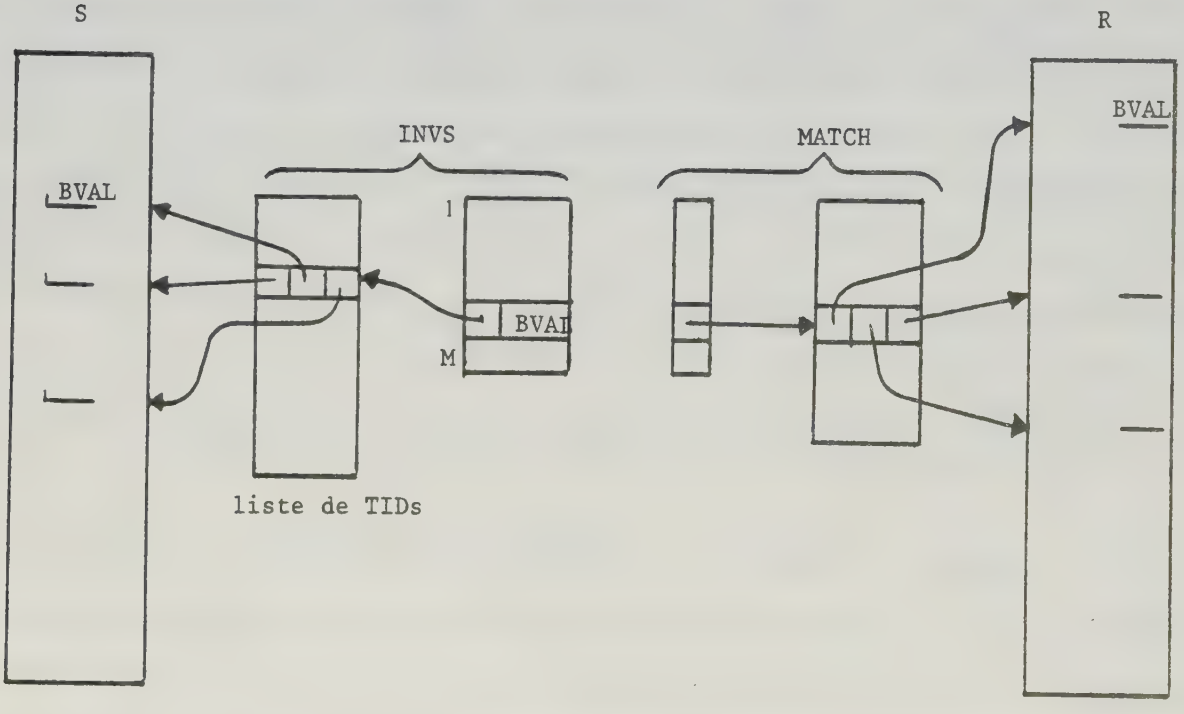


figure 8 : organisation des fichiers inversés

```
procedure JOINDRE2 (R,S)

    1) pour chaque r de R
       faire
         2) TROUVER L dans INV-S tel que
              INV-S(L).BVAL=r.A
            si L existe,ranger l'adresse de r dans MATCH(L)

    3) pour j=1,M  faire
         -LIRE les listes INV-S(j).BLIST et MATCH(j)
         -pour chaque ad. tuple ds INV-S(j).BLIST
           faire
             pour chaque ad. tuple ds MATCH(j)
               faire
                  AJOUTER ces adresses à la liste résultat
```

figure 9 : algorithme avec fichier inversé.

L'étape 1 (lecture de R) demande m accès MS. Pendant l'étape 2,l'adresse de r est écrite dans la liste du fichier inversé MATCH. Des que le tampon contenant une partie de cette liste est plein (J éléments ont été écrits),il est écrit en MS. L'ensemble de ces tampons (de capacité M*J)sera ainsi écrit n.El/M*J fois. L'étape 3 (lecture des listes des fichiers inversés) coûte 2M acces MS: à chaque pas,une liste supposée tenir en mémoire est lue. Il reste à lire les tuples adressés par les paires d'adresses récupérées dans la liste résultat,et deux cas sont à distinguer selon que l'index a la propriété de "groupant" ou non.

1) L'index est groupant signifie que les tuples sont placés en MS par regroupement suivant la valeur d'attribut de jointure, ainsi les tuples peuvent être lus en blocs pour un nombre d'accès de (nj + mj)/K.

2) L'index est non groupant signifie que les tuples doivent être accédés chacun en un accès,par un nombre d'accès total de mj + nj. Ce terme rend coûteux cette méthode,lorsque la relation résultat est grande. Le nombre total d'entrées-sorties est finalement:

$$t(9) = m + m.El/M*J + 2M + (nj + mj)/L$$

Où L = K si l'index est groupant et L = 1 sinon.

La taille mémoire utilisée par cette méthode est :K+M*J.

Il est à remarquer que cette procédure peut être adaptée pour pallier aux limites de la procédure JOINDRE1 décrite en 3-1,simplement en ajoutant la création dynamique d'un fichier inversé moyennant un coût supplémentaire de n + n.E2/M*J évalué dans (GOTL75) à 25% du cout t(9).

4-2. Algorithme d'INGRES

Lors de la décomposition d'une question multivariable, Ingres réalise la jointure, qui porte sur deux variables, en substituant une variable par les tuples de la relation correspondante, puis en générant autant de questions monovariables que de tuples. La substitution d'une variable peut être vue comme un parcours séquentiel de la relation correspondante. Pour chaque tuple ,une sous-question est générée pour accéder à l'autre relation. Dans le cas qui nous intéresse, il existe un index haché ou ISAM sur l'attribut de jointure de la deuxième relation. Dans l'algorithme, l'index est accédé pour tout tuple de la première relation. Si cet accès est valide (le tuple appartient à la jointure),le balayage de cet index entre les mêmes valeurs de clés (pour les synonymes) fournit les adresses des tuples,qu'il suffit d'accéder directement pour composer la jointure. L'index haché n'est utilisé que pour l'équi-jointure. Figure 11 est représenté un shéma de l'algorithme d'INGRES. Cet algorithme est décrit par la figure 10.

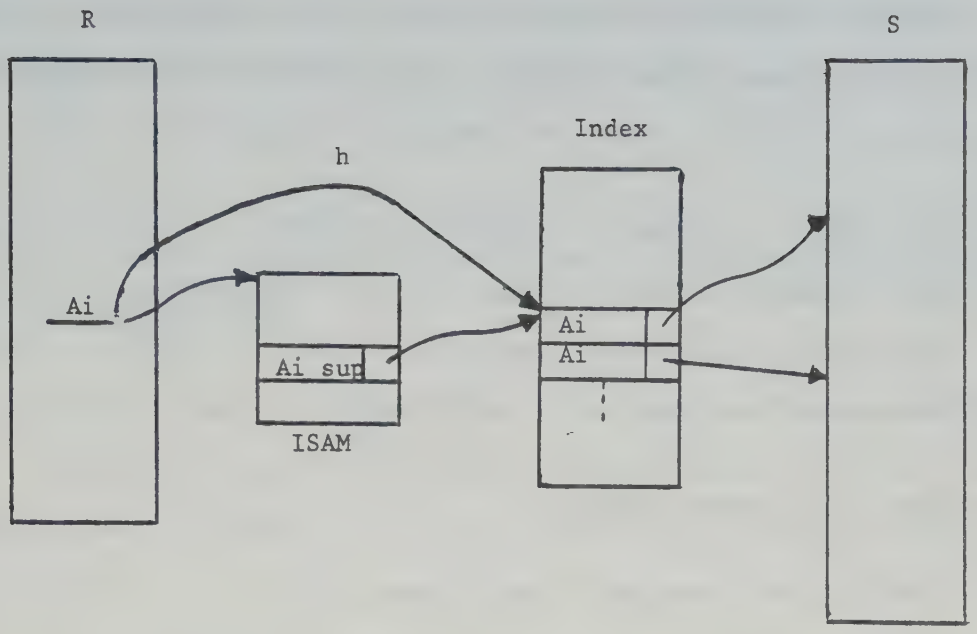


figure 10 : jointure avec index haché ou ISAM par INGRES

```
algorithme

1) BALAYER la plus petite relation
   pour chaque tuple
     si il existe un index haché et c'est une équijointure
       alors
         - ACCEDER à cet index
       sinon
         - ACCEDER à l'index ISAM
     2) BALAYAGE de l'index pour les mêmes valeurs de clé
     et accès direct aux tuples pour réaliser la jointure
```

figure 11 : algorithme d'Ingres avec index.

Pour simplifier l'analyse et les formules, les accès aux pages de débordement des index sont ignorés. L'étape 1 exige m entrées-sorties pour lire la première relation. Nous évaluons maintenant le coût d'accès à l'index. La probabilité pour que la page de l'index référencée soit déjà en mémoire centrale n'est pas négligeable, et importe lorsque la taille (I) de l'index est faible. Cette probabilité

est 1/I. Le nombre d'entrées-sorties réalisées pour chaque tuple pour accéder à l'index haché est donc (1-1/I), soit pour toute la relation m*El*(1-1/I). Il faut encore nj accès directs aux tuples de la deuxième relation qui participent à la jointure. Le coût est finalement : m + m*El*(1-1/I) + nj .

Avec un index ISAM, il faut un accès à l'index primaire plus un accès par niveau d'index secondaire par tuple. En supposant en moyenne deux niveaux ,le nombre d'accès à l'index est 3m*El*(1-1/I) . Dans ce cas ,le coût en entrées-sorties est : m + 3m*El*(1-1/I) + nj.

4-3. Algorithme de SystemR (BLAS77)

System-R utilise deux méthodes avec index.

4-3-1. S'il existe des index sur les attributs de jointure

Cette méthode emploie un parcours des deux index pour déterminer les couples de tuples qui ont la même valeur d'attribut de jointure. Tout tuple r de R est accédé par valeur d'attribut de jointure de l'index. S'il satisfait à la restriction,l'index de S est parcouru pour cette valeur d'attribut de jointure. Chaque tuple s de S correspondant est accédé et s'il satisfait à la restriction ,le couple (r,s) ainsi trouvé est projeté et concaténé dans la relation résultat. L'algorithme est représenté par la figure 12.

```
- BALAYAGE simultané des index
1) par le 1er,ACCES aux tuples pour obtenir
   le prochain répondant à la restriction

2) par le 2ème,ACCES aux tuples de même attribut de jointure
   répondant à la restriction
     pour chaque paire trouvée
       faire la projection dans la relation cible

- répéter les étapes 1 et 2 jusqu'à la fin des index
```

figure 12 : algorithme avec index sur attributs de jointure.

Les balayages des index requièrent 2*I acces MS,I étant la taille d'un index en nombre de pages.Le coût de la lecture des tuples est ,comme en 4-1,soumis à la propriété de l'index: groupant ou non. Le nombre total d'acces MS égale donc 2I + (nj + mj)/L ,
Où L = K si l'index est groupant et L = 1 sinon.

4-3-2. S'il existe des index sur attributs de jointure et attributs de restriction.

C'est un cas très favorable. Il est facile de diminuer le stockage des données intermédiaires grâce à l'utilisation des adresses de tuples notées TIDs. L'accès aux index permet d'obtenir les TIDs des tuples qui satisfont la restriction sur R et S. Ceux-ci sont triés et rangés dans les fichiers W1 et W2, comportant respectivement n1 et n2 pages (n1 et n2 étant tres petits). Le parcours des index sur les attributs de jointure permet d'obtenir les paires de TIDs dont les tuples participent à la jointure. Pour chaque paire ainsi trouvée,un accès à W1 et W2 permet de vérifier si elle satisfait la restriction. Cet algorithme est représenté figure 13.

algorithme utilisant les TIDs

1) par les index sur attributs de restriction.
   - OBTENIR les TIDs des tuples répondant a la restriction
   - TRIER dans W1 et W2

2) BALAYAGE simultané des index sur attributs de jointure pour obtenir les paires de TIDs qui se joignent

   - VERIFIER pour chaque paire (TID1,TID2) :
     TID1 est dans W1 et TID2 est dans W2
     si oui LIRE les tuples et PROJETER dans la relation cible

figure 13 : algorithme avec index sur attributs de jointure et de restriction.

Les balayages des index exigent 4*I acces MS. Le tri des fichiers temporaires W1 et W2 exige 2*n1 * log(n1) + 2*n2 * log(n2) accès,en

supposant que W1 et W2 ne tiennent pas en mémoire et n1+n2 acces sont encore nécessaires pour vérifier que les tuples appartiennent bien à la jointure. Le coût pour accéder les tuples adressés par les paires (TID1,TID2) est (nj + mj) et le même qu'en 4-1 et 4-3-1. Le nombre d'accès MS total est alors:

$$t(13) = 4I + 2n1*\log n1 + 2n2*\log n2 + n1 + n2 + (mj+nj)/L.$$

La taille mémoire est de 4 pages plus 2 tampons pour W1 et W2.

5. CAS AVEC CLES

Il est très frequent que l'un ou les deux attributs de jointure soient clés primaires. Quand un attribut est clé,une solution simple consiste à balayer la relation dont l'attribut de jointure n'est pas clé et pour chaque tuple, balayer l'autre relation en appliquant une recherche dichotomique jusqu'à trouver LE tuple qui se joint. Cette solution donne des résultats de l'ordre de m + m*K*log n acces MS. Deux autres méthodes sont proposées dans le cas où il y a un index sur l'attribut de jointure pour une relation ,ceci étant fortement probable.

5-1. Algorithme d'INGRES

Ingres utilise des fichiers ISAM et cet algorithme consiste simplement à une lecture séquentielle de la plus grande relation et à accéder directement les tuples de l'autre qui participent à la jointure. Cet algorithme est décrit figure 14.

```
1) BALAYER la 1ère relation
   2) pour chaque tuple
        - ACCES DIRECT à la 2ème relation par clé
        - si "accès valide"  appliquer la jointure
```

figure 14 : algorithme d'Ingres.

Cette méthode exige m acces à la première relation, 2m*E1*(1 - 2/I)

accès à l'index du fichier ISAM (supposé à 2 niveaux),et nj accès aux tuples,ce qui fait au total: m + 2m*El*(1 - 2/I) + nj accès MS.

5-2. Algorithme 3 de Gotlieb

Lorsqu'il existe un fichier inversé trié sur une relation et que les attributs de jointure sont des clés,une adaptation de la procedure JOINDRE2 est possible. Les balayages des listes d'adresses du fichier inversé peuvent être diminuées car,à une valeur d'attribut de jointure clé ne correspond qu'une seule adresse de tuple. Ceci permet de lire des blocs du fichier inversé contenant des éléments de la forme (valeur,indice du tuple ayant cette valeur de clé) et de faire ainsi un partitionnement de la relation R,une page étant représentée par les tuples de valeurs d'attributs de jointure comprises entre les débuts et fins de valeurs de blocs du fichier inversé. Pour ce faire,un tableau contenant les valeurs de clés en début de blocs du fichier inversé est initialisé,ce qui nécessite un balayage du fichier inversé de plus que la procedure JOINDRE2.

Ainsi,l'étape 2 de création du fichier inversé sur R,voit le nombre de comparaisons diminuer et la jointure est faite en lisant deux blocs de chaque fichier inversé et pour chaque élément (valeur,adresse) a accéder aux tuples. Le nombre d'entrées-sorties de cet algorithme est le même que celui de la procedure JOINDRE2,augmenté du cout de création du tableau RANGE, contenant les valeurs de clés en début de blocs, qui nécessite la lecture du fichier inversé (M acces MS) et fait que cette méthode est 25% fois plus lente que la procedure JOINDRE2 et exige un nombre d'acces MS de:

$$m + m.El\ /M*J + 3M + (nj + mj)/L$$

Cette méthode ne profite pas bien du fait que les attributs de jointure soient clés et s'avère bien moins éfficace que la méthode

simple avec recherche par dichotomie.

6. COMPARAISONS

Dans cette section,un tableau récapitulatif est présenté figure 15 puis des comparaisons sont faites des coûts en nombre d'entrées-sorties des méthodes les plus représentatives qui ont été décrites. Le tableau page suivante récapitule donc les coûts d'accès MS pour chaque méthode.

Il est peu facile de comparer une opération réalisée par des systèmes différents, indépendamment du contexte dans lequel elle s'applique. Nous nous proposons donc de comparer ces méthodes avec les hypothèses simplificatrices énoncées et en fixant certains paramètres, pour montrer leur influence. Les comparaisons sont séparées en deux cas selon l'absence ou la présence d'index. Le cas où les attributs de jointure sont des clés primaires est considéré comme un facteur d'optimisation et est rapproché du cas avec index. Le nombre d'entrées-sorties est évalué en fonction de la cardinalité des relations et est représenté par un graphe. Les paramètres intervenant également pour chaque relation,leur cardinalité est supposée être la même,ceci pour simplifier. Le coût de la jointure,isolée des opérations de restrictions et projections est seul pris en compte.

figure 15 : tableau récapitulatif des méthodes présentées

	pas de chemins d'accès	index	attributs de jointure clés
GOTLIEB	alg.1 : $m + m.n$ alg.2 modifié : $n + n.E2/M.J + m + m.E1/M.J + 2M + nj + mj$	alg.2 : $m + m.E1/M.J + 2M + (nj + mj)/L$	alg.3 : $m + m.E1/M.J + 3M + (nj + mj)/L$
INGRES	avec fichier haché $n + (n.E2)^2 /N + m + mj$	1) index haché $m + m.E1.(1-1/I) + nj$ 2) index ISAM $m + 3m.E1.(1-3/I) + nj$	fichier ISAM $m + 2m.E1.(1-2/I) + nj$
SYSTEM-R	1) tri $2\ m.Log\ m + 2\ n.Log\ n + m + n$ 2) multipasse $p\ (m+n)$	1) Index sur att. join $2.I + (nj + mj)/L$ 2) index sur att. rest. et att. join $2.I + 2\ n1\ log\ n1 + 2\ n2\ log\ n2 + n1 + n2 + (nj + mj)/L$	
CAFS	avec hachage (et 2 semijointures) $2m + 2n + 2\ nj/K.log(nj/K) + 2\ mj/K.log\ (mj/K) + (nj + mj)/K$		L = K si index groupant L = 1 sinon

6-1. Comparaison des méthodes sans index

Les méthodes suivantes sont comparées:

1 : l'algorithme 2 de Gotlieb (création dynamique de deux fichiers inversés)

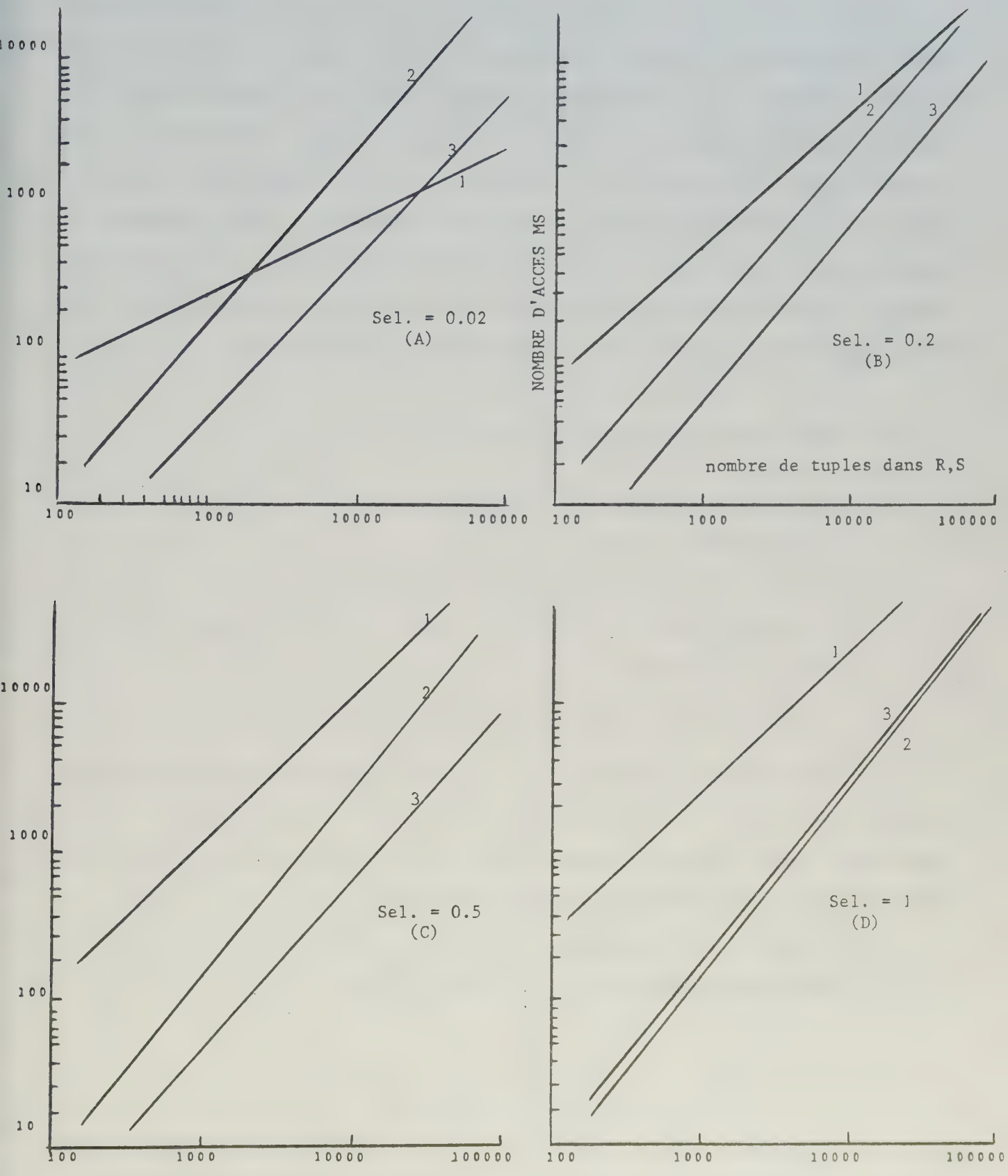
2 : l'algorithme avec tri de SYSTEM-R

3 : l'algorithme de CAFS (en faisant deux semijointures)

La méthode multipasse présente des performances comparables à la méthode de tri et n'est applicable que pour une petite relation cible , c'est pourquoi la méthode avec tri est choisie. La procedure inspirée d'Ingres a aussi été écartée car elle n'est pas comparable avec les méthodes sans index, mais plutôt avec les méthodes avec index,le fichier haché étant considéré alors comme un chemin d'accès particulier. Le coût de création du fichier haché rendrait cette méthode inférieure aux autres,alors que cette opération,dans le système INGRES,a lieu lors des restrictions et projections.

Figure 16, quatre situtions,dépendant chacune des coefficients de sélectivité de la jointure,sont décrites et montrent l'influence de ce parametre. Par exemple,la situation A décrit le nombre d'accès nécessaires en fonction de la taille des relation lorsque 2% des tuples de chaque relation participent à la jointure. Les autres parametres (taille mémoire) sont fixés ainsi : I=M=n/100, J=10, K=100, E1=E2=100.

figure 16 : comparaison des méthodes sans index.



Observations

L'algorithme avec création de fichiers inversés est généralement le moins performant et son coût augmente à mesure que la relation cible devient grande, i.e. lorsque le coefficient de selectivité de la jointure devient grand. Le nombre d'accès aux tuples appartenant a la jointure est en effet mj + nj, car il faut un accès pour obtenir un tuple. L'algorithme utilisant le tri est indépendant du paramètre de sélectivité de la jointure et donne de bons résultats;il est le meilleur lorsque 100% des tuples participent à la jointure,cas relativement rare. Géneralement,l'algorithme inspiré de CAFS est le meilleur .

6-2. Comparaison des méthodes avec index

Les méthodes suivantes sont comparées:

1 : l'algorithme 2 de Gotlieb,

2 : l'algorithme d'Ingres avec un index ISAM,

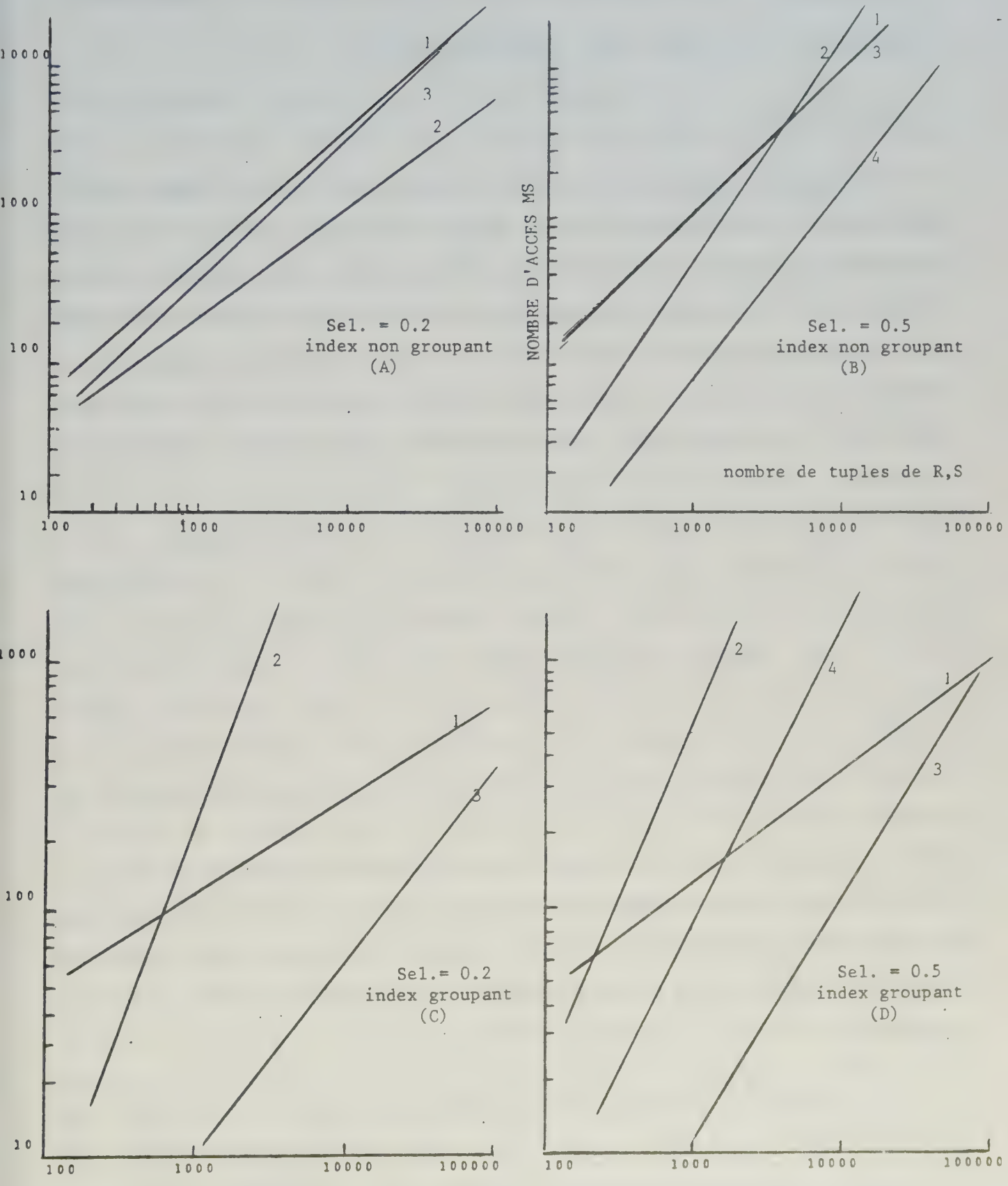
3 : l'algorithme de System-R lorqu'il existe des index sur attributs de jointure.

4 : la méthode avec semi-jointures (sans index)

Il a été discuté plus haut de l'intérêt du placement des tuples sur mémoire secondaire regroupés sur l'attribut de jointure, facteur important pour les performances des méthodes avec index. Les comparaisons seront donc faites dans deux cas:

- l'index est non groupant ,
- l'index est groupant ,

figure 17 : comparaison des méthodes avec index.



La figure 17,pour chacun des cas,présente deux situations montrant l'influence de la sélectivité de la jointure pour les valeurs 0.2 et 0.5,ce qui est suffisant pour les observations.

Observations

Il apparait clairement que la propriété de l'index détermine la performance des méthodes avec index. Lorsque l'index n'est pas groupant,les méthodes sans index,représentées ici par la courbe 4,donne de meilleurs résultats puisque la méthode consistant à balayer l'index puis accéder aux tuples aléatoirement ne permet pas de lire par blocs,et revient donc plus cher. Il est important de remarquer qu'un seul index peut avoir cette propriété avantageuse d'être groupant,en reflétant le placement des tuples sur mémoire secondaire,ce qui limite considérablement l'intéret de telles méthodes lorsque la taille de la relation résultat est élevée. Ingres ne pratique pas de placement par groupement et sa méthode de jointure donne de mauvais résultats. Ceci est du aux nombreux accès aux index mais aussi à l'hypothese m=n car dans le cas où m est très inférieur à n, cette méthode devient performante. Par exemple, si le nombre de tuples sélectionnés dans la première relation est inférieur au nombre de clauses acceptées dans une question monovariable, le coût en entrées-sorties devient de l'ordre de m+n. La méthode de Gotlieb parait un peu plus coûteuse que celles de System R car elle ne suppose l'existence que d'un index,alors que celle de System R en suppose deux. Une conclusion est qu'a une situation donnée caractérisée par les valeurs de ces paramètres ,une méthode est optimale.

Il est aussi à remarquer que le coût de maintenance des index lors des opérations de mise a jour n'a pas été pris en compte. Ceci désavantage les méthodes par tri ou hachage lors des comparaisons.

7. CONCLUSION.

Dans ce chapitre, plusieurs méthodes pour traiter les jointures ont été considérées. Un premier critère d'éfficacité est la diminution du nombre d'entrées- sorties sur mémoire secondaire. Cet objectif est réalisé en utilisant suffisamment d'espace de travail ,en organisant les fichiers externes et en utilisant des chemins d'accès sur les attributs qui sont fréquemment joints (index,liens entre relations). Le meilleur moyen pour réaliser éfficacement la jointure est de réduire si possible la jointure de grandes relations stockées en mémoire secondaire à la jointure de plus petites relations en mémoire centrale. Ainsi,une jointure est généralement précédée des opérations de restrictions et projections.

La représentation des relations peut rendre la jointure plus éfficace si,par exemple,elle est en forme "divisée",i.e. les attributs étant stockés séparément,ce qui élimine des lectures d'attributs inutiles et , ainsi , réduit les temps d'entrées-sorties et le temps de traitement. Mais, lorsque tous les attributs des tuples sont demandés, le nombre d'accès augmente considérablement. Le choix de la représentation dépend donc du type d'applications à supporter.

Il en ressort trois groupes de méthodes évitant le produit cartésien des deux relations ,qui a le coût prohibitif de m+mn accès MS. Le premier groupe basé sur le tri donne de bons résultats de l'ordre de 2m *log(m) +2n *log(n)+m+n . Le deuxième avec index est performant lorsque l'index a la propriété d'être "groupant" et permet les lectures des tuples participant à la jointure par blocs. Lorsque l'index n'est pas groupant et que la proportion de tuples participant à la jointure est grande, la méthode par tri s'avère plus efficace. Le troisième groupe utilisant le hachage donne des résultats de l'ordre de a(m+n) ,où

a est une constante. Il est à noter que le coût de mise à jour des index a été omis, car dificile a évaluer, ce qui défavorise quelque peu les méthodes par tri ou hachage lors des comparaisons.

En conclusion,à une jointure donnée caractérisée par les paramètres suivants : taille des relations,selectivité du prédicat de jointure, présence d'index,nature du prédicat de jointure (équi-jointure ou non) correspond une méthode optimale pour la traiter.

CHAPITRE III :

ALGORITHMES DE JOINTURE MULTIPROCESSEURS

1. INTRODUCTION

De nombreuses machines bases de données sont actuellement en cours de développement (AUER80, BANE78, DEWI79, STON79, WAH80). Leur objectif essentiel est de pouvoir répondre à des questions complexes, exprimées a l'aide de langages basés sur la logique de prédicats, portant sur des grandes bases de données relationnelles avec de meilleurs temps de réponse que les systèmes plus classiques (ASTR76,STON76). Ces machines sont généralement réalisées à l'aide d'un ensemble de processeurs permettant l'exécution en parallèle d'une requête. Elles sont également le résultat d'une approche architecturale qui essaie de distribuer la puissance des traitements plus près des périphériques sur lesquels les données sont stockées et ainsi de décharger les fonctions de traitement des données des processeurs principaux (LANG79).

Un des facteurs limitatifs essentiels à la réalisations de systèmes relationnels est l'opération de jointure de relations. Plusieurs algorithmes de jointure monoprocesseurs ont été présentés et discutés au chapitre II. Dans un tel contexte sans parallélisme et en l'absence d'index, pour des relations de cardinalité n, une méthode simple consistant a filtrer le produit cartésien des relations donne un temps d'exécution proportionnel à n**2. Un meilleur groupe d'algorithmes basés sur le tri (BLAS77) permet d'obtenir un temps d'exécution de l'ordre de n*log n. Un troisième groupe, basé sur le hachage , permet de réduire encore ce coût et d'obtenir un temps d'exécution de l'ordre de n, c'est à dire linéairement proportionnel au volume des opérandes. Cependant, cette dernière méthode ne permet que le traitement des semi-jointures (BERN79a), cas particulier de jointure où les attributs résultats n'appartiennent qu'à une relation opérande.

Ce chapitre présente des extensions de ces algorithmes à un contexte multiprocesseur et analyse leurs performances. L'objectif

idéal est de diviser les temps d'exécution de ces algorithmes par le nombre de processeurs traitants. Au paragraphe 2, le modèle d'architecture de base à l'étude des algorithmes est précisé. Il est dérivé de la machine bases de données SABRE, introduite au chapitre I. Les éléments d'architecture invoqués lors du traitement des jointures sont un ensemble de processeurs de filtrage associés aux unités de disques, une mémoire cache, et un ensemble de processeurs de jointure. Ces processeurs possèdent une mémoire locale pour stocker les données pendant le traitement. Ils communiquent par l'intermédiaire d'un réseau d'interconnexion. Ensuite, les critères d'analyse sont donnés au paragraphe 3. Ce sont les paramètres de performance nécessaires pour évaluer les couts d'exécution (entré-sortie, traitement CPU, transmission de messages) des algorithmes. L'hypothèse de base est que les relations opérandes sont trop grandes pour être contenues en mémoire cache ou dans les mémoires propres des processeurs.

Puis, trois algorithmes de jointure multiprocesseurs sont présentés paragraphes 4, 5 et 6. L'algorithme utilisant le produit cartésien des relations (BORA80) consiste à répartir une relation (appelée externe) sur un ensemble de processeurs et à diffuser la seconde relation (appelée interne), de sorte que chaque processeur joigne une partie de la relation externe avec la relation interne entière. Ceci permet de profiter au mieux du degré de parallélisme introduit. L'algorithme de jointure par tri est ensuite présenté. Il emploie un tri ascendant multiprocesseur des deux relations sur l'attribut de jointure suivi d'une fusion monoprocesseur pour réaliser l'opération. Le paragraphe suivant est consacré à l'étude d'un algorithme avec hachage. Il consiste à hacher la plus petite relation sur l'attribut de jointure en mémoire cache et, simultanément, à mettre à jour un tableau de bits (BABB79) pour chaque valeur d'attribut de jointure différente. La plus grande relation est alors répartie avec le tableau de bits sur

l'ensemble des processeurs de jointure. Le fichier haché est alors accédé pour chaque tuple de la relation répartie tel qu'un bit correspondant dans le tableau soit marqué. Finalement,au paragraphe 7,les temps nécéssaires à l'exécution d'une jointure selon l'algorithme choisi sont comparés. Les résultats sont en particulier fonction du nombre de processeurs utilisés, de la taille des relations opérandes, et de la proportion de tuples participant à la jointure. Il est finalement montré que, pour une configuration donnée, chaque algorithme a un domaine d'application défini par les caractéristiques des relations opérandes et résultat.

2. MODELE D'ARCHITECTURE

L'environnement dans lequel les algorithmes de jointure sont étudiés est la machine bases de données SABRE (GARD81). Pour une description plus détaillée, le lecteur peut se reporter au chapitre 2. La machine est composée d'un ensemble de processeurs connectés au moyen d'un réseau d'interconnexion. Les processeurs échangent des commandes directement et les données au travers d'une mémoire cache. Ci-dessous,nous décrivons sous forme d'un modele simplifié a trois niveaux les éléments essentiels de l'architecture qui permettent de réaliser des jointures.

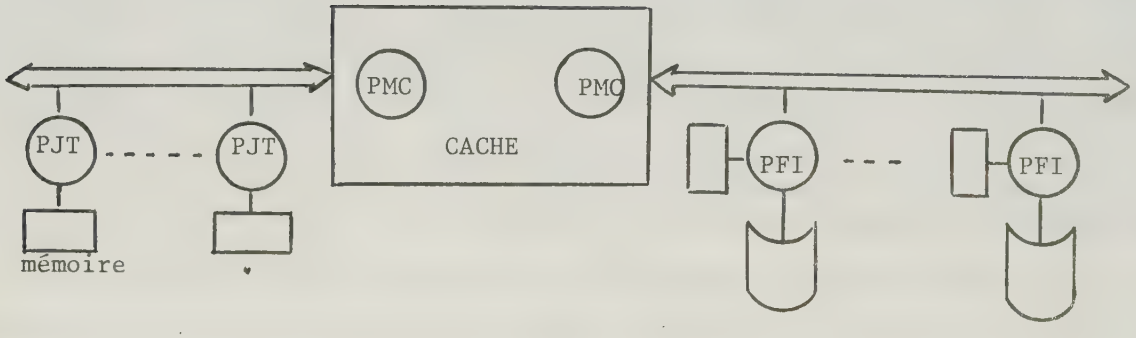
Le premier niveau de l'architecture est un ensemble de processeurs de filtrage de données (BANC80,FAUD80). Ces processeurs travaillent chacun en parallèle avec un flot de données provenant d'un disque. La stratégie de placement garantit que les relations puissent être lues en parallèle, grâce à un partitionnent des relations sur l'ensemble des unités de disques. Il a été montré dans (BORA81) que, si les lectures parallèles de disques étaient employées, l'approche processeur par disque donnait les meilleurs résultats, car elle évite la saturation du

canal. Ces processeurs assurent la sélection "au vol" des données du flot répondant à une expression de sélection et le transfert des données retenues vers la mémoire cache. Une expression de sélection est composée d'une suite de références déterminant les attributs à projeter et d'une condition de sélection. Une condition de sélection est une conjonction (et) de disjonctions (ou) de clauses de la forme (Ai op C),où Ai est une référence d'attribut,op un des opérateurs de relation $\leq, <, =, \neq, >, \geq$ et C une constante. Les filtres qui seront développés (FAUD80) sont des processeurs relativement simples très proches de la forme générale des automates. Ils permettent de réaliser un traitement synchrone au flot, ou en d'autres termes effectuent une selection a la vitesse de rotation des disques dans une page de mémoire secondaire correspondant généralement à une piste. Les filtres communiquent avec la mémoire cache au moyen d'un bus tres rapide, de type multibus (INTE79).

Les données sélectionnées par les processeurs de filtrage sont stockées dans la mémoire cache divisée en pages, qui constitue le deuxième niveau. Le gérant de la mémoire cache a pour role l'anticipation des lectures de pages et met en oeuvre un algorithme de remplacement des pages les moins récemment utilisées, pour les vider sur disque. Les filtres sont capables de travailler de la même manière sur des données stockées dans des pages en mémoire cache ou en mémoire secondaire. Ceci est utile pour opérer des sélections sur des données déja en mémoire cache.

Le troisième niveau est un ensemble de processeurs spécialisés de jointure chargés d'exécuter les opérations de jointure et tri. sélections effectuées sur des relations. Chaque processeur de jointure dispose d'une mémoire propre de plusieurs pages. Il communique avec la mémoire cache pour acquérir ou restituer des pages lors de l'exécution

d'une opération. Les processeurs de jointure sont connectés avec la mémoire cache par un réseau d'interconnexion spécifique. Pour obtenir des performances optimales, ce réseau de connexion doit avoir deux propriétés. La première est que plusieurs processeurs de jointure puissent lire ou écrire simultanément des pages en mémoire cache. La seconde est la possibilité de diffuser la même page à un nombre quelconque de processeurs de jointure en un seul transfert. Ce type de réseau est faisable (BRAT80, DEWI79), et différentes alternatives seront étudiées dans le projet SABRE. L'objet de cette thèse n'est pas d'y choisir une solution, car elle affectera les performances de tout le système. Pour notre propos, nous supposerons un réseau d'interconnexion ayant ces deux propriétés, n'introduisant pas de délai significatif. Ces éléments d'architecture sont décrits figure 1.



PJT : processeur de jointure et tri
PMC : processeur de la mémoire cache
PFI : processeur de filtrage

figure 1 : Eléments d'architecture utilisés

3. PARAMETRES ET CRITERES D'ANALYSE

Afin de décrire différents algorithmes de jointure ,nous supposerons que nous avons à effectuer la jointure de deux relations R et S dans la relation résultat T. Les paramètres suivants seront utilisés pour l'évaluation des différents algorithmes proposés :

m : nombre de pages de R,

n : nombre de pages de S,

C : taille de la mémoire cache utilisable pour la jointure (en nombre de pages),

p : nombre de processeurs de jointure,

b+1 : taille de la mémoire propre d'un processeur de jointure (en nombre de pages) = b pages d'entrée + 1 page de résultat,

t : nombre moyen de tuples par page de chaque relation, supposé le même pour R et S,

SJ : coefficient de sélectivité de la jointure défini par :

nombre de pages de T / (m*n) ,

SR : coefficient de sélectivité de la semi-jointure de R par S défini par :

nombre de pages résultant de la semi-jointure de R par S / m ,

SS : coefficient de sélectivité de la semi-jointure de S par R défini par :

nombre de pages résultant de la semi-jointure de S par R / n ,

Lors de la description détaillée des algorithmes,les processeurs seront supposés être numérotés de 0 à p-1, les pages de la relation R de 0 à m-1 et celles de S de 0 à n-1.

L'analyse des performances de nos algorithmes requiert une définition rigoureuse des paramètres de performances de base. Ces paramètres doivent mesurer le coût d'entrée-sortie, le coût de

traitement et le coût de communication pour exécuter un algorithme dans un système à architecture multiprocesseur. Le temps d'exécution d'un algorithme à prendre en compte pour les comparaisons sera exprimé en termes de ces coûts de base. Les paramètres identifiés peuvent prendre différentes valeurs suivant les caractéristiques matérielles de l'architecture développée. L'hypothèse principale est que les données sont échangées et traitées par unités de pages. D'autre part, les relations traitées sont supposées suffisamment grandes pour ne pouvoir être entièrement contenues dans les mémoires locales des processeurs et (ou) en mémoire cache. L'opération de jointure est étudiée sans tenir compte des opérations de restriction-projection. Enfin, pour simplifier l'analyse, il est supposé que la base de données est stockée sur une seule unité de disque, avec un seul processeur de filtrage. Les paramètres de base à l'évaluation des performances sont maintenant détaillés.

3-1. Cout d'entrée-sortie :

Dans une architecture multiprocesseur où des processeurs échangent des données,la lecture ou l'écriture d'une page par un processeur se fait par l'intermédiaire de la mémoire cache et peut avoir deux valeurs, selon que la page est présente en mémoire cache ou réside sur une unité de mémoire secondaire. Le coût de transfert d'une page de mémoire cache vers un processeur ou entre deux processeurs est noté Tcp. Le coût de transfert d'une page de mémoire secondaire vers la mémoire cache est noté Tdc.

Il nous est alors utile de définir le coût moyen de lecture d'une page par un processeur en tenant compte de la probabilité que la page demandée soit déja en mémoire cache. Le rôle important du gérant de la mémoire cache est notamment le maintien de cette probabilité forte en anticipant les lectures de pages d'une même relation, ceci étant

facilité par le fait que, généralement une relation référencée sera lue entièrement. Cette probabilité est assimilée au taux d'éfficacité de la mémoire cache et notée F. Le coût moyen de lecture d'une page par un processeur est défini par CL :

$$CL = F*Tcp + (1-F)*(Tcp+Tdc)$$

De la même manière, nous pouvons évaluer le coût moyen d'écriture d'une page, en introduisant la probabilité qu'il y ait une page libre en mémoire, noté F'. Cette probabilité doit être maintenue faible pour une bonne occupation de la mémoire cache et relativement constante , en développant un algorithme de remplacement des pages les moins récemment utilisées. Le coût moyen d'écriture d'une page par un processeur est défini par CE :

$$CE = F'*Tcp + (1-F')*(Tcp+Tdc)$$

3-2. Coût de communication

Les transferts de pages sont vus comme des opérations d'entrée-sortie, aussi le coût de communication à prendre en compte se limitera aux messages de demande de pages et aux messages de réponse. Nous pouvons donc ajouter le coût de ces messages M aux coûts d'entrée-sortie définis précédemment et remplacer CL par CL + M(demande) + M(réponse) et CE par CE + M(demande) + M(réponse). Les messages de contrôle tels que ceux nécessaires à l'allocation de processeurs sont en nombre négligeable par rapport aux messages de demande de page et réponse, et seront ignorés pour la clarté de l'analyse.

3-3. Coût de traitement CPU

Un certain nombre de paramètres permettent d'identifier les traitements éffectués par les processeurs lors de l'exécution parallèle d'un algorithme. Tout d'abord, V représente le cout d'une opération simple telle que comparer deux attributs ou calculer des fonctions de

hachage. Le coût pour transférer un tuple dans une zone mémoire est défini par I. Nous pouvons alors définir les coûts de traitement suivants :

coût de jointure de deux pages : selon l'algorithme, les pages a joindre peuvent être triées ou non. Si les pages ne sont pas triées, deux manières de faire sont possibles. La première consiste à trier les pages sur l'attribut de jointure puis à les fusionner. La seconde, plus simple, consiste à comparer tout tuple d'une page avec tout tuple de l'autre et est meilleure que la première, car elle ne nécéssite que des opérations de comparaison, ce qui est moins couteux que les transferts de tuples exigés par le tri interne. Ainsi, le coût pour joindre deux pages non triées , défini par CJ0 , est : CJ0 = t**2*V

Le cout pour joindre deux pages triées par une opération de type fusion est défini par : CJ1 = 2*t*V

coût pour trier une page : Le coût moyen de tri interne d'une page de t tuples requiert $t*\log_2 t$ opérations de comparaisons et transferts (KNUT73). Il est défini par :

$$CT = t*\log_2 t*(V+I)$$

cout de fusion de q pages : La fusion de q pages triées exige q*t opérations de comparaisons et transferts. En comptant le lecture et écriture de q pages, le coût de fusion de q pages est défini par :

$$CFq = q*t*(V+I) + q*(CL + CE)$$

4. JOINTURE PAR PRODUIT DES RELATIONS

4-1. Description de l'algorithme

Cet algorithme est une version parallèle de l'algorithme monoprocesseur de jointure le plus inefficace, puisqu'il fait le produit cartésien des relations. Cet algorithme simple s'adapte particulièrement bien à l'exécution parallele. La méthode consiste à joindre chaque page d'une relation avec l'autre relation entière. Cet algorithme a été décrit et évalué dans (BORA80) pour des processeurs disposant de trois pages de mémoire locale, où deux tampons sont utilisés pour entrer des pages et un tampon pour sortir des pages. La méthode est maintenant généralisée pour supporter des tailles de mémoires propres de plus de trois pages.

L'exécution de cet algorithme par p processeurs en parallèle disposant chacun d'une mémoire propre de b+1 pages peut être réalisée ainsi. La plus petite relation est choisie comme externe et est séquentiellement répartie par unités de (b-1) pages entre les p processeurs. Ensuite, la seconde relation (interne) est diffusée page par page aux p processeurs. Pour chaque page de la relation interne, chaque processeur réalise la jointure interne avec (b-1) pages de la relation externe , en utilisant un tampon résultat d'une page.

Il est à remarquer que, comme tout tuple d'une relation sera comparé à tout tuple de l'autre, cet algorithme peut traiter indifféremment tout type de jointure (equi-jointure, inequi-jointure). Si la relation externe ne peut être contenue dans les mémoires locales de p processeurs, le même procédé doit être répété pour les pages suivantes de la relation externe.

L'algorithme exécuté par un processeur k est plus précisément représenté figure 2 sous la forme d'une procédure JOINDREP (R,S,F), où R

et S sont respectivement les relations externe et interne et E l'expression de sélection définissant la jointure. Sup(a) désigne le premier entier supérieur ou égal à a. BR et BS sont les tampons du processeur de jointure k de tailles respectivement (b-1) pages et 1 page. BJOINDRE (BR,BS,RES,E) est une procédure qui effectue la jointure des deux tampons BR et BS et écrit le résultat dans un tampon RES d'une page. ECRIRE (RES) permet d'écrire le tampon RES en mémoire cache.

```
Procedure JOINDREP (R,S,E);

   pour i=1 à Sup(m/(p*(b-1))) faire
     r0 = (i-1)*p*(b-1) + k*(b-1);
     lire (b-1) pages de R dans BR à partir de r0;
     pour j=1 à n  faire
        s0 = j-1;
        obtenir 1 page de S dans BS à partir de s0;
        BJOINDRE (BR,BS,RES,E);
        si "RES plein"  ECRIRE (RES)

fin-JOINDREP ;
```

figure 2 : Algorithme de jointure par produit cartésien

4-2. Evaluation de l'algorithme

Si la relation externe ,supposée R, peut être mémorisée par les p processeurs, c'est à dire m=p*(b-1), la boucle externe de l'algorithme est effectuée une fois. Chaque processeur lit en parallèle (b-1) pages de R. Le temps d'exécution de l'algorithme est alors :

C(P) = C(lire (b-1) pages de R) + n* C(lire une page de S et BJOINDRE)

Si la relation externe ne peut être contenue dans les mémoires propres des processeurs, la boucle interne doit être répétée m/(p*(b-1)) fois puisqu'elle est imbriquée dans la boucle externe. Le coût de jointure de (b-1) pages avec une page est (b-1)*CJ0. Le coût d'écriture

du résultat de la jointure de deux pages est dépendant du coefficient de sélectivité de la jointure SJ, et, en comptant les transferts de tuples résultats plus l'écriture en mémoire cache, est SJ*((t*I) + CE). Le temps d'exécution de l'algorithme par produit (P) est alors :

$$C(P) = m/(p*(b-1)) * (CL + n*(CL + (b-1)*CJ0 + SJ*((t*I)+CE)))$$

Ce coût est proportionnel à m/p + m*n/p, et montre que l'on a intérêt à minimiser la valeur de m en choisissant la plus petite relation comme externe. La formule précédente prouve aussi que le degré de parallélisme est constant (i.e. la charge de chaque processeur est constante) pendant toute l'exécution de l'algorithme. Il est a noter cependant que la dernière passe peut utiliser un nombre plus réduit de processeurs si le rapport m/(p*(b-1)) ne donne pas un résultat entier.

5. JOINTURE PAR TRI DES RELATIONS

5-1. Méthode de tri utilisée

L'algorithme par tri des relations emploie un tri multiprocesseur des deux relations sur les attributs de jointure suivi d'une fusion permettant de réaliser l'opération de jointure. Nous considérons l'equi-jointure. L'algorithme de tri est une version parallèle de la méthode ascendante dite fusion, dont la description monoprocesseur a été faite dans (KNUT73, p.247). Dans (BORA80), un algorithme de jointure, utilisant un tri-fusion parallele où les processeurs sont organisés en arbre binaire (PREP78) est présenté. L'analyse des performances de cet algorithme montre qu'il est généralement moins efficace que l'algorithme de jointure par produit, lorsque le nombre de processeurs est élevé. Ceci est du au fait que, après une certaine phase, le degré de parallélisme est divisé par deux à chaque étape de fusion, de telle sorte qu'a la dernière étape de fusion, un seul processeur fusionne la

relation entière. Ci-dessous, nous proposons et évaluons un algorithme utilisant un tri-fusion d'ordre b plus performant dans un contexte multiprocesseur. En outre, si b=p la dernière étape de fusion se fait en une seule passe.

5-2. Description de l'algorithme

Nous rappelons d'abord le principe de l'algorithme de tri-fusion d'ordre b monoprocesseur (KNUT73, p.247). Supposons avoir à trier n éléments. Une monotonie est définie comme une séquence ordonnée d'éléments, donc l'ensemble à trier contient n monotonies de un élément. La méthode consiste à successivement fusionner b monotonies de K éléments en une monotonie triée de K*b éléments, en partant de K=1. Pour chaque étape i, chaque ensemble de b monotonies de b**(i-1) éléments est fusionné en une monotonie de b**i éléments. A partir de i=1, le nombre d'étapes nécessaires pour trier n éléments est $\log_b n$ et le nombre de comparaisons est $n*\log_b n$ (KNUT73).

L'application de cet algorithme dans une machine bases de données possédant une architecture multiprocesseur de type indiqué au paragraphe 2 peut être effectué comme suit. Supposons avoir à trier une relation de n pages suffisamment grande pour ne pas tenir en mémoire cache. Rappelons que chaque processeur possède une mémoire locale de (b+1) pages, où b tampons sont utilisés comme pages d'entrée et un tampon comme page de sortie. A chaque étape i, chaque processeur fusionne un ensemble de b monotonies de b**(i-1) pages en une monotonie triée de b**i pages. La fusion de b monotonies est faite en lisant successivement une page nécessaire de chaque monotonie dans b tampons d'entrée et en transférant les tuples ordonnés dans un tampon de sortie, écrit en mémoire cache lorsqu'il est plein. selon le critère de tri, dans un tampon résultat de 1 page étendu en mémoire cache, en demandant les lectures de pages suivantes nécessaires. Les pages lues lors de la

première étape sont d'abord triées en mémoire. Nous supposerons pour simplifier l'analyse que modulo(n,p) = 0, ou en d'autres termes que p proceseurs divisent la relation à trier en parties égales et se partagent exactement la même charge. A chaque étape de fusion, le nombre de monotonies à fusionner est divisé par b alors que la taille de chacune est multipliée par b. Pour une étape, soit M le nombre de monotonies à fusionner, si M est plus grand que p*b, il faut M/(p*b) passes pour faire cette étape de fusion. Dès que ce nombre M est réduit à p*b, il ne faut qu'une passe pour fusionner toutes les monotonies de taille M/(p*b) et cette phase est dite optimale (DEWI80). La phase optimale engendre donc p monotonies de n/p pages chacune et si l'on choisit p=b, un seul processeur peut les fusionner en une seule étape. Mais comme dans certaines configurations , p peut être supérieur à b, la solution pour réaliser les dernières fusions est d'organiser les processeurs en arbre d'ordre b lors de la dernière phase appelée post-optimale, où le nombre de processeurs nécessaires à chaque étape est divisé par b. A la dernière étape, un seul processeur fusionne la relation entière. Le nombre d'étapes de la phase post-optimale est alors $\log_b p$. Un exemple d'application de cette méthode est décrit figure 3.

La jointure de deux relations triées est réalisée par une opération monoprocesseur de type fusion. L'adaptation de cet algorithme a l'inequi-jointure modifie seulement l'opération de fusion, qui devient plus complexe et coûteuse.

L'algorithme exécuté par le processeur k pour trier la relation R sur le critère E est représenté figure 5 par la procedure TRIER (R,E). Cette procedure comporte deux boucles correspondant aux phases sous-optimales et post-optimales. Elle appelle la procedure locale PASTRI décrite figure 4 qui réalise une étape de tri d'ordre b des

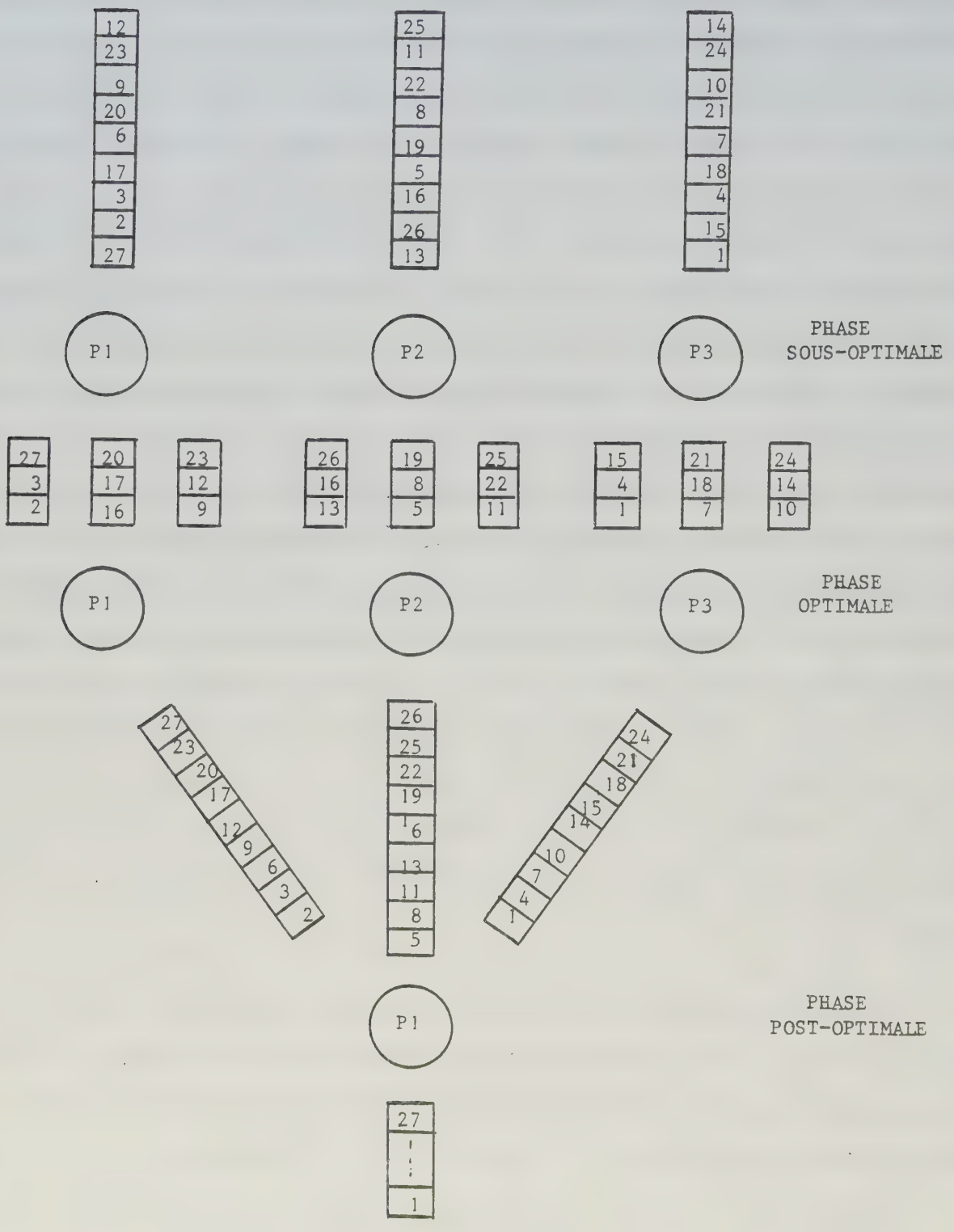


figure 3 : tri de 27 pages avec 3 processeurs
(p=3 , b=3)

monotonies de taille M. La procedure LIRE (R(e),BR(f)), où e=0,m-1 et f=0,b-1 permet de lire la page R(e) dans le tampon BR(f) en mémoire propre. La procedure TRIERI (BR(i)) réalise le tri interne de la page BR(i). La procedure INSERER (r,RES) insere le tuple r dans le tampon RES. La procedure ECRIRE (RES,R(i)) permet l'écriture de la page RES dans la page i de la relation R en mémoire cache. L'algorithme exécuté lors de la phase sous-optimale par le processeur k peut être vu comme une fusion monoprocesseur d'ordre b sur la partie de relation comportant les pages k*n/p à ((k+1)*n/p) -1.

```
procedure  PASTRI (j,M,passel) ;

   $ j : no de la lère page à traiter,
     M : taille de monotonie (en nombre de pages),
     passel : booleen $

k' =k
pour 1=0 à b-1
   faire
     LIRE(R(j+1*(M*b**(i-1))), BR(1));
     si i=1 et passel  faire TRIERI (BR(1)) ;
FIN-MONO = "faux";
tant que FIN-MONO = "faux" faire
   r="le tuple dans BR(x) à écrire"
   INSERER (r,RES);
   si "RES plein "  faire
       ECRIRE (RES, R(k'))
       k' = k+1;
   si r = "dernier tuple de BR(x)" alors
       si BR(x) = "dernière page de la monotonie x"
        alors
           "indiquer fin monotonie x"
           si "toutes monotonies traitées"
             alors  FIN-MONO ="vrai"
        sinon
           "lire la page suivante de la monotonie x"
fin-PASTRI.
```

figure 4 : algorithme réalisant une étape de tri d'ordre b.

```
procedure TRIER (R,E);
   $ R : relation , E : expression du critère de tri $

pour i =1 à Sup (log_b(n/p))          $ phase sous-optimale $
  faire
      PASTRI (k*(n/p),1,vrai)

i=0 ;
p' = Sup (p/b) ;
tant que p'=1 ou k p'-1  $phase post-optimale $
     i=i+1;
     PASTRI (k*b*(n/p) , n/p, faux);
     p' =p'/b;

fin-TRIER
```

figure 5 : algorithme de tri fusion d'ordre b.

5-3. Evaluation de l'algorithme de tri

Nous évaluons le temps d'exécution du tri d'une relation de n pages. A la phase optimale correspondant à l'étape K, chaque processeur crée une monotonie de taille b**K pages, et comme chacun des p processeurs a traité exactement n/p pages, l'équation suivante s'en déduit :

$$b^{**}k=n/p, \text{ où } K=\log_b n/p.$$

Comme nous comptons les étapes à partir de 1, K est le nombre d'étapes jusqu'à la phase optimale. Lors des étapes précédant la phase optimale, chaque processeur est employé et réalise exactement n/(p*b) opérations de fusions de b pages, en opérant un tri interne a la première passe. Le coût de la phase sous-optimale est alors :

$$(n/p)*CT + \log_b(n/p)* (n/(b*p))* CFb$$

La phase optimale engendre p monotonies de n/p pages chacune. Il faut $\log_b p$ étapes pour fusionner p monotonies où, à chaque étape, le nombre de processeurs employés est divisé par b, et la charge de chaque processeur multipliée par b. Premièrement, p/b processeurs réalisent n/p fusions de b pages, puis p/(b**2) processeurs en font b*n/p et ainsi

de suite jusqu'à ce qu'un processeur n'en réalise plus qu'exactement n/b. Le coût de cette dernière phase est alors :

$$\underbrace{(n/p + b*n/p + (b**2)*n/p + \ldots + n/b)}_{\log_b p \text{ termes}} * CFb$$

$$= ((1- b^{**\log_b p}) / (1-b)) * n/p * CFb$$

Le cout de tri d'une relation de n pages est donc :

$$C(tri(n)) = (n/p)*CT + \log_b(n/p)*(n/(b*p))*CFb + ((1-b^{**\log_b p})/(1-b))*(n/p)*CFb$$

5-4. Evaluation de l'algorithme de jointure par tri

Le coût d'exécution de la jointure est égal au coût de tri de la relation R, plus le coût de tri de la relation S, plus le coût de fusion. L'opération de type fusion est réalisée par un processeur qui lit les deux relations triées, les joint et écrit la relation résultat. Le cout pour fusionner m et n pages est :

$$C(fusion(m,n)) = (m+n)*CL + \max(m,n)*CJJ + m*n*SJ*CE$$

Finalement, le coût d'exécution de la jointure par tri (T) est :

$$C(T) = C(tri(n)) + C(tri(m)) + C(fusion(m,n))$$

6. ALGORITHME PAR HACHAGE D'UNE RELATION

6-1. Description de l'algorithme

La méthode proposée utilise une technique de hachage et des tableaux de bits. L'utilisation de tableaux de bits pour implanter l'opération de semi-jointure a été décrite par Babb (BABB79). L'idée, très riche, est de hacher l'attribut de jointure et de se servir du résultat comme adresse dans un tableau de bit. La présence d'un bit marqué dans le tableau signifie que des tuples participant a la jointure existent. La propriété de cette technique est d'éliminer la plupart des

données inutiles au résultat de l'opération de jointure. Afin de supporter la jointure comme la semi-jointure, la méthode est améliorée et adaptée à un contexte multiprocesseur. Nous considérons l'equi-jointure. La méthode procède essentiellement en deux phases, avant lesquelles un tableau de bits B est initialisé en mémoire cache. Lors de la première phase, la plus petite relation est lue en mémoire cache et hachée sur l'attribut de jointure par le gérant du cache de sorte que chaque tuple soit écrit dans un paquet d'un fichier aléatoire, géré aussi en mémoire cache. Simultanément, pour chaque valeur d'attribut de jointure v, B(h(v)) est marqué (mis a 1), où h désigne la fonction de hachage appliquée à l'attribut de jointure. La première phase se termine lorsque la relation entière a été hachée. Lors de la deuxième phase, le tableau de bits est diffusé à p processeurs et la plus grande relation est séquentiellement répartie sur p processeurs. Chaque processeur a besoin d'une mémoire locale d'au moins 4 pages, ou deux sont utilisées comme tampons d'entrée, une comme tampon de sortie et une autre pour contenir le tableau de bits. Chaque processeur reçoit une page de la plus grande relation et procède au traitement suivant. Pour chaque tuple de la page tel que sa valeur d'attribut de jointure v' vérifie B(h(v')) = 1, un paquet du fichier aléatoire est directement accédé en précisant la clé v' pour retrouver le ou les tuples se joignant. Comme un paquet du fichier aléatoire peut contenir plus d'une page, le paquet est lu page par page. Chaque tuple de la page lue du paquet est aors comparée avec v' pour compléter la jointure.

Cette méthode fait un usage important du hachage. Pour être applicable, le nombre de valeurs distinctes d'attributs de jointure dans la relation hachée doit être supérieur au nombre de paquets du fichier aléatoire et la distribution de ces valeurs relativement uniforme. A la première phase, le choix de hacher la plus petite relation est fait pour minimiser la création et le coût de stockage du fichier haché.

Le problème classique avec le hachage est les collisions . Si v1 et v2 sont deux valeurs d'attributs de jointure différentes, nous pouvons avoir h(v1) = h(v2). Ces collisions conduisent à des accès inutiles au fichier aléatoire lors de la lecture de la deuxième relation. Afin de réduire ces accès inutiles,sans toutefois les éliminer, une solution possible consiste à utiliser plusieurs fonctions de hachage h1,h2,...hq associées chacune à un tableau de bits B1,B2,...Bq.

Si la fonction de hachage du fichier aléatoire a la propriété de maintenir l'ordre, i.e. soient deux valeurs d'attributs v1 et v2, si v1 op v2 alors h(v1) op h(v2), où op est un opérateur d'inégalité, l'algorithme peut supporter l'inequi-jointure.

Figure 6, nous présentons l'algorithme exécuté par un processeur travaillant sur la mémoire cache avec q fonctions de hachage et tableaux de bits sous forme d'une procedure HACHER. Cette procedure crée un fichier aléatoire à partir de la relation R sur l'attribut de jointure A. Ce fichier est particulier en ce sens que les paquets peuvent être de taille variable en nombre de pages et les pages relatives a un même paquet sont chainées, ce qui évite la gestion d'une zone de débordement. Nous supposons initialement ces tableaux de bits B1,B2,...Bq mis a zéro. R est la plus petite relation,S la plus grande et E est l'expression de sélection définissant la jointure. L'algorithme de jointure exécuté ensuite par un processeur k est décrit par la procedure JOINDREH figure 7. A et B sont les attributs de R et S comparés. BR désigne la mémoire privée d'un processeur de jointure. INSERER (r,a) réalise l'insertion du tuple r dans le paquet a du fichier aléatoire. h est la fonction de hachage utilisée pour le fichier aléatoire alors que h1,h2,...hq sont celles utilisées pour les tableaux de bits B1,B2,...Bq. LIRE (a,M) lit le paquet a du fichier aléatoire en mémoire cache dans le tampon M.

TJOINDRE (r,s,RES) est une procedure qui effectue la jointure des tuples r et s et ajoute le résultat dans le tampon RES.

```
procedure HACHER (R,A) ;

   pour chaque page de R
     pour chaque tuple r faire
        B1(h1(r.A))=1;...Bq(hq(r.A))=1;
        INSERER (r,h(r.A));
        si "page h(r.A) pleine"  ECRIRE(h(r.A);
fin-HACHER
```

figure 6 : Algorithme de hachage d'une relation

```
procedure JOINDREH (R,S,E);

   pour j=1 à S(n/p) faire
     s0 =(j-1)*p + k;
     lire 1 page de S dans BR à partir de s0;
     pour chaque tuple s de BR faire
         si B1(h1(s.B))=1 et ... et Bq(hq(s.B))=1
            alors  LIRE (h(s.B),M);
              pour chaque tuple r de M faire
                  si s.B=r.A  alors
                     TJOINDRE(r,s,RES)
                     si "RES plein"  alors ECRIRE (RES);
fin-JOINDREH;
```

figure 7 : Algorithme de jointure par hachage

6-2. Evaluation de l'algorithme

Le cout d'exécution de l'algorithme est le coût d'exécution de la procedure HACHER plus le coût d'exécution parallèle de la procedure JOINDREH . Le coût de diffusion des tableaux de bits est négligé (un seul transfert). Nous rappelons que C pages sont disponibles en mémoire cache pour l'opération de jointure. La création du fichier haché en mémoire cache (à partir de R) utilise C pages, et consiste à créer m paquets si $m \leq C$, ou C paquets sinon. Dans le premier cas, le fichier peut être maintenu en mémoire cache pendant l'exécution entière de l'opération. Dans le dernier cas, les pages d'un même paquet seraient

liées et écrites sur disque, et retrouvées en utilisant une table des addresses physiques. Le coût de création du fichier haché est alors : m*(C(lire R) + hacher) + (m-C)*C(ecrire R). La lecture d'une page de R en tenant compte du facteur F coûte (1-F)*Tdc. Le coût de hachage d'une page de t tuples est t*(V+I), et le coût d'écriture, en tenant compte de la probabilité qu'il ait des pages libres en plus des C pages réservées à l'algorithme est (m-C)*(1-F')*Tdc. Le coût du hachage d'une relation de m pages est alors :

$$C(\text{hacher}(m)) = m*((1-F)*Tdc) + (t*(V+I)) + (m-C)*(1-F')*Tdc$$

Le coût d'excution de la procedure JOINDREH comporte la lecture de la relation S en parallèle par p processeurs, les accès au fichier haché et l'écriture de la relation résultat. Chaque processeur lit n/p pages de la relation S et, pour chacun des t tuples, accède le tableau de bits, pour le coût de (n/p)*(CL+(t*V)). L'accès au fichier aléatoire est nécéssaire pour chaque tuple de S participant à la jointure. Le nombre de tuples par page de S participant à la jointure est déterminé par le coefficient de sélectivité de la semi-jointure de S par R défini par SS. Chaque paquet du fichier aléatoire contient m/C pages. Le coût d'accès au fichier aléatoire exige pour chaque page de S lue (t*SS*(m/C)*CL et ceci est répété (n/p) fois pour la relation S entière. Le coût d'écriture de la relation résultat de taille m*n*SJ par p processeurs en parallèle est (m*n*SJ)/p *CE. Finalement, le coût d'exécution de l'algorithme par hachage (noté H) est :

$$C(H) = m*(((1-F)*Tdc)+(t*(V+I))) + (m-C)*(1-F')*Tdc + (n/p)*((CL+V) + (t*SS*(m/C)*CL) + (m*SJ*CE))$$

7. COMPARAISONS

A partir des formules développées précédemment, les algorithmes sont analysés. Les couts comparés sont :

$$C(P) = m/(p*(b-1)) *(CL +n*(CL + (b-1)*CJ0 + SJ*((t*I)+CE)))$$

$$C(T) = (m/p)*CT + ((\log_b (m/p)*m/(b*p)) + (1-b^{**}\log_b p)*(1-b)) *(n/p)*CFb$$

$$+ (n/p)*CT + ((\log_b (n/p)*n/(b*p)) + (1-b^{**}\log_b p)*(1-b)) *(n/p)*CFb$$

$$+ (m+n)*CL + \max(m,n)*CJ1 + m*n*SJ*CE$$

$$C(H) = m* (((1-F)*Tdc) +(t*(V+I))) + (m-C)*(1-F')*Tdc$$

$$+(n/p)* ((CL+V) + (t*SS*(m/C)*CL) + (m*SJ*CE))$$

Les hypothèses concernant les possibilités des processeurs, paramétrés par les coûts de base définis au paragraphe 2 sont décrites en annexe 1.

7-1. Comparaisons en fonction de la taille des relations

La figure 8 illustre le comportement des algorithmes en fonction de la taille des relations, supposées composées du même nombre de pages (m=n). Une configuration comportant 8 processeurs de jointure avant chacun une mémoire propre de 5 pages avec 16 pages de mémoire cache allouée à la jointure sous-tend ces résultats. Trois courbes décrivent les performances de l'algorithme par hachage pour trois coefficients de sélectivité de semi-jointure (0.5, 0.1, 0.01). Le coefficient de sélectivité de jointure est supposé être 0.01, ce qui est moyen. Il apparait que l'algorithme par tri se comporte mieux que les algorithmes par produit cartésien et par hachage (quand $SS \geqslant 0.1$) pour des relations de tailles élevées. Le comportement de l'algorithme par hachage dépend essentiellement de SS, le coefficient de sélectivité de semi-jointure . Lorsque peu de tuples de la relation répartie sur p processeurs participent a la jointure ($SS \leqslant 0.01$), le nombre d'acces au fichier aléatoire est faible et cet algorithme donne les meilleurs résultats.

Tant que SS est inférieur à 0.1 pour la configuration fixée, l'algorithme par hachage est toujours meilleur que l'algorithme par produit. Au dela de 0.5, il devient non performant (courbe H0). La taille mémoire cache allouée à la jointure (C) influe ausi de manière significative sur l'algorithme par hachage, puisque si C est suffisamment grand la relation hachée peut tenir en mémoire.

La figure 9 illustre dans le plan (m,n) la limite de performance de l'algorithme par produit par rapport a l'algorithme par tri, pour la meme configuration que précédemment. De la même maniere, la figure 10 représente la limite de l'algorithme par tri par rapport a l'algorithme par hachage (pour SS=0.1)), l'algorithme par produit n'étant pas représenté car, dans ce cas, il est moins intéressant que les deux autres.

7-2. Comparaison en fonction de la configuration

La configuration est essentiellement caractérisée par le nombre de processeurs de jointure. La taille de la mémoire cache est aussi un paramètre important. Les figures 11 et 12 illustrent le comportement des algorithmes en fonction du nombre de processeurs pour des tailles de relations petites (figure 11) et plus grandes (figure 12). L'algorithme par produit se comporte en 1/p alors que l'algorithme par tri évolue en $1/\log_2 p$. Pour des configurations à nombre de processeurs élevés,l'algorithme par produit est plus intéressant que l'algorithme par tri. Avec un coefficient de sélectivité de semi-jointure moyen (0.1), l'algorithme par hachage donne de bons résultats lorsque le nombre de processeurs est grand. C'est parce que le coût de la seconde phase de l'algorithme est de complexité 1/p. Généralement, ce coefficient est faible , ce qui rend l'algorithme par hachage très attrayant. Il est aussi remarquable que, pour l'algorithme par tri, au

dela d'un grand nombre de processeurs, l'adjonction de nouveaux processeurs diminue peu le temps d'exécution. Ceci est du a l'augmentation du nombre d'étapes pour réaliser la phase post-optimale du tri, qui est proportionnelle au nombre de processeurs. En fait, l'algorithme par tri est meilleur pour les configurations telles que p=b, où la phase post-optimale ne nécessite qu'une seule étape. Lorsque p devient très supérieur à b, la phase post-optimale dégrade le degré de parallélisme.

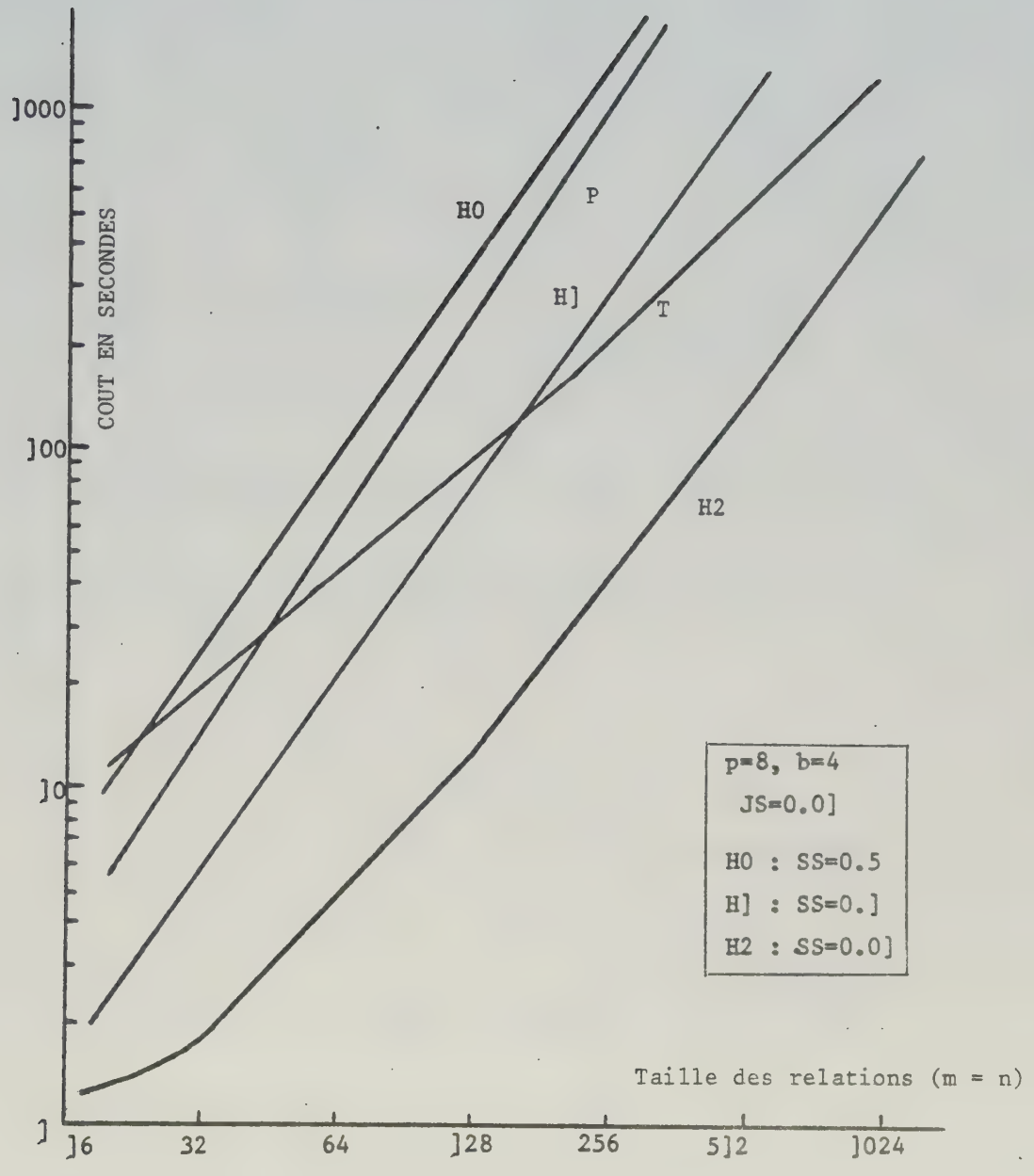


figure 8 : Coûts d'exécution des algorithmes
en fonction des tailles des relations

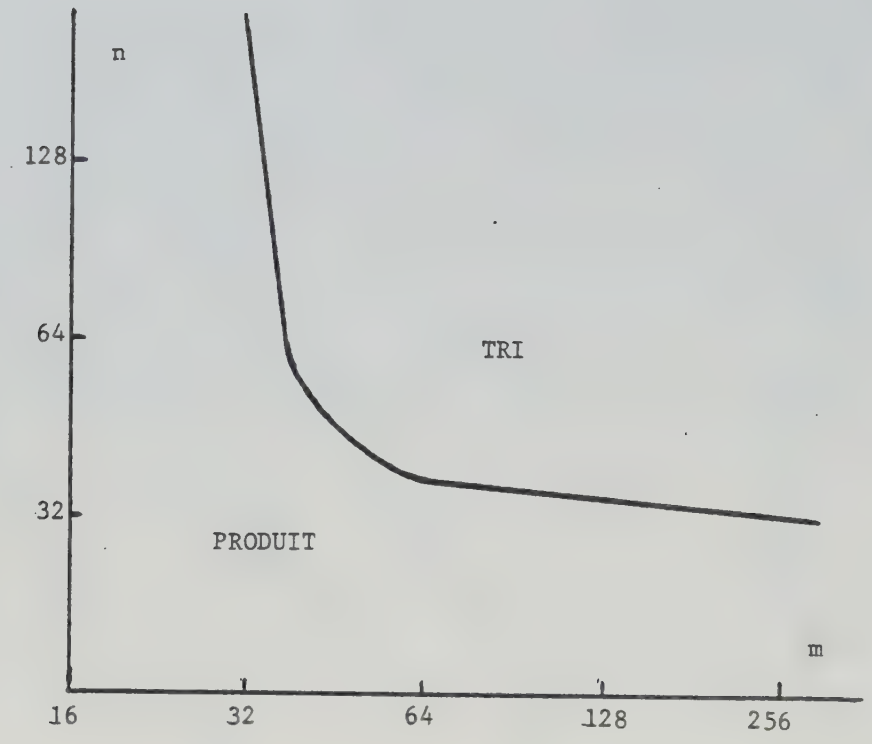


figure 9 : Limite de l'algorithme par produit par rapport a l'algorithme par tri (p=8,b=4,SJ=0.01)

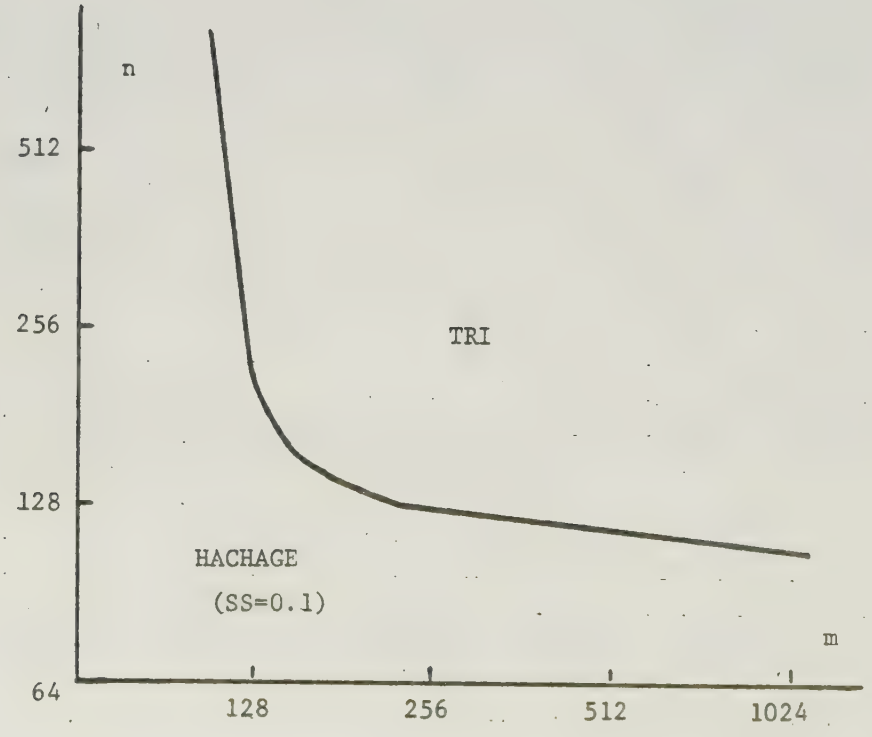


figure 10 : Limite de l'algorithme par hachage par rapport a l'algorithme par tri

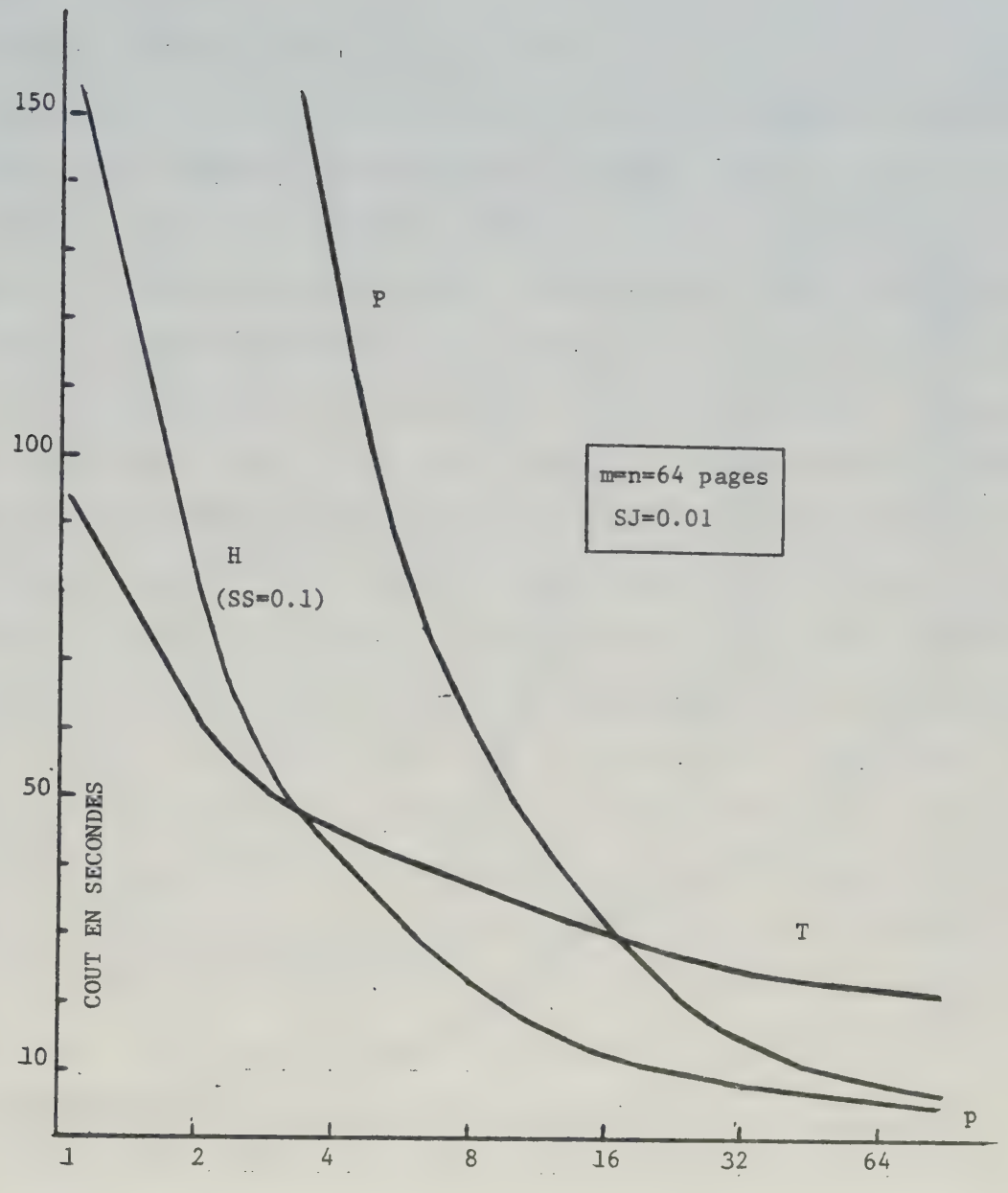


figure 11 : Coûts d'exécution des algorithmes en fonction du nombre de processeurs (p)

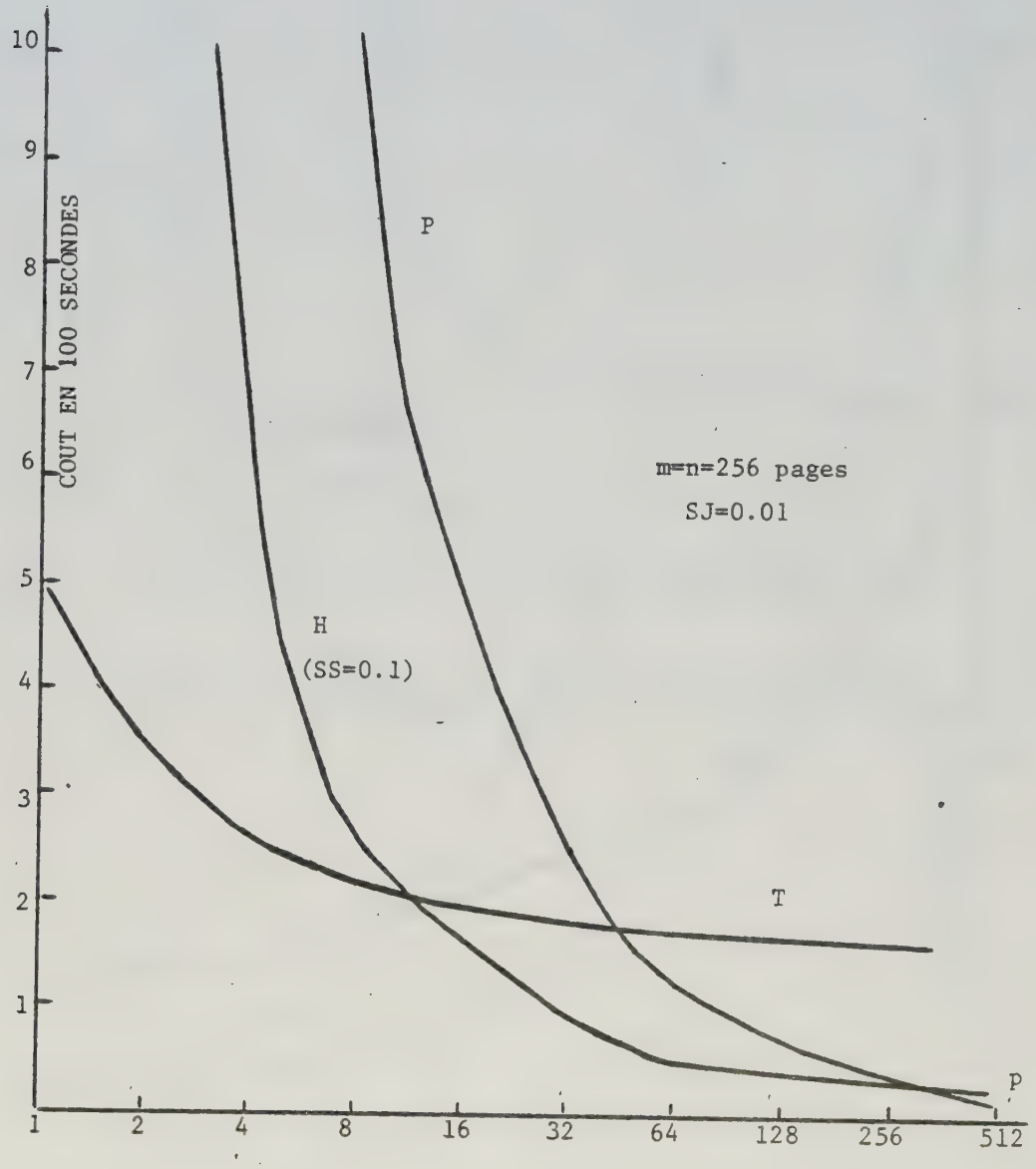


figure 12 : Coûts d'exécution des algorithmes en fonction du nombre de processeurs

8. CONCLUSION

Dans ce chapitre, trois algorithmes pour réaliser les jointures dans une machine bases de données ont été présentés et analysés. Auparavant, les éléments d'architecture utiles pour les jointures dans la machine bases de données SABRE, et les paramètres de coûts d'exécution (entrées-sorties, CPU, communication) ont été détaillés. Puis, trois algorithmes ont été présentés et comparés en fonction de leurs coûts d'exécution. L'algorithme par produit est le plus simple. Il est également celui qui se comporte le mieux quand le nombre de processeurs exécutant en parallèle la jointure devient élevé. De plus, comme chaque tuple est comparé avec chaque autre tuple, il réalise facilement l'inequi-jointure. L'algorithme par tri plus complexe est performant relativement aux autres quand la taille des relations devient élevée. Pour une configuration à nombre de processeurs élevé, l'addition de nouveaux processeurs n'implique pas une grande augmentation de performances. Cependant, cet algorithme a l'avantage de produire la relation résultat triée sur l'attribut de jointure, et aussi d'utiliser un opérateur nécessaire à la machine (tri). L'algorithme par hachage est meilleur lorsque la proportion de tuples de la plus grande relation participant a la jointure est peu élevée. C'est dans ce cas que le dispositif tableau de bits permet d'éviter des accès inutiles au fichier aléatoire. En conclusion,il est à souligner qu'aucun algorithme ne peut être choisi lors de la conception de la machine, indépendamment de la connaissance des relations à joindre. Une machine bases de données bien spécifiée devrait donc implémenter plusieurs types d'algorithmes,et,à chaque jointure,choisir le meilleur algorithme a utiliser,en calculant pour chacun d'eux le temps d'exécution nécessaire. Ces algorithmes pourraient être implantés sur des microprocesseurs commercialisés, sans exiger de matériel spécial et coûteux.

Les résultats présentés peuvent être étendus dans plusieurs directions. En premier lieu, les algorithmes présentés peuvent être optimisés. Par exemple, la phase post-optimale de l'algorithme de tri pourrait être améliorée en trouvant une autre méthode. Deuxièmement, d'autres méthodes basées sur les index ou des tables représentant des jointures pré-évaluées pourraient être utilisées. Les performances de ces méthodes dépendraient essentiellement de la stratégie de placement des relations sur les unités de disques. La stratégie de placement par regroupement multi-attribut (KARL81) d'améliorer encore le cout d'exécution des jointures. Une autre considération est que, généralement, l'opération de jointure met en présence de grandes relations alors que la relation résultat est petite. C'est parce qu'une faible proportion de tuples de chaque relation participe au résultat final. Comme le coût de communication dans une machine à architecture répartie est important, il pourrait être diminué en utilisant un opérateur cher aux concepteurs de systèmes de bases de données réparties : la semi-jointure (BERN79a). L'étude d'algorithmes de semi-jointure et leur intérêt dans une machine bases de données fera l'objet du chapitre IV.

CHAPITRE IV :

ALGORITHMES DE SEMI-JOINTURE DE RELATIONS

POUR SYSTEMES MULTIPROCESSEURS

1. INTRODUCTION

La semi-jointure est un opérateur relationnel qui a la propriété de réduire le coût de traitement des questions invoquant des opérations binaires. Cet opérateur procède en ne retenant de la base que les données utiles à la question et diminue ainsi le volume des relations opérandes. Le facteur de coût dominant dans les systèmes de gestion de bases de données réparties (LEBI80, ROTH80) est le coût de communication inter-sites, proportionnel au volume de données transférées. Afin de réduire le coût de transmission lors du traitement de questions réparties, ces systèmes font un usage extensif de l'opérateur relationnel de semi-jointure pour traiter les opérations binaires portant sur des relations stockées à des sites différents. Certaines machines bases de données (BABB79,OZKA77) supportent aussi cet opérateur pour minimiser les transferts avec l'ordinateur hôte lors des opérations binaires. Dans une machine bases de données à architecture répartie (GARD81), le coût de communication inter-processeurs est un facteur important à prendre en compte, et l'adjonction d'un opérateur avant la propriété de réduire le volume des données échangées devient nécessaire. De plus, l'intégration possible d'une telle machine dans un contexte de bases de données réparties par sa connexion à un réseau confirme ce besoin.

La semi-jointure est une opération binaire de l'algebre relationnelle. La semi-jointure de la relation R par la relation S selon les attributs A de R et B de S est le sous-ensemble de tuples de R

$$\{ r \in R / \; s \in S : (r.A \; \Theta \; s.B) \}$$

où Θ est un des opérateurs $\leq, <, =, \neq, >, \geq$. Ainsi, la semi-jointure (de R par S) est equivalente à une jointure (de R et S) et une projection (sur les attributs de R). C'est aussi un cas particulier et fréquent de jointure où les attributs résultats demandés dans une question ne

portent que sur une relation. Cette opération est beaucoup moins chère que la jointure et a un coût linéairement proportionnel au volume des relations opérandes et résultat (BABB79). Il a été montré au chapitre II que, en l'abscence d'index, la méthode monoprocesseur consistant à remplacer la jointure de deux relations par la jointure de leurs semi-jointures donnait généralement les meilleurs résultats.

De récents travaux s'intéressent à la théorie des semi-jointures. Le rôle important des semi-jointures dans le problème d'optimisation de questions en base de données réparties a été reconnu dans (BERN79a). Dans cette étude, chaque question est supposée définie par une conjonction de clauses d'equi-jointures. Il y est démontré en particulier que , pour une classe importante de questions appelées "questions arborescentes", il existe toujours des séquences d'equi-semi-jointures minimales pour répondre à la question. Ces séquences sont appelées réducteurs complets et un algorithme pour les construire y est présenté. Pour les autres questions, appelées "questions cycliques", il est prouvé qu'il n'existe pas de réducteur complet. Partant de ces résultats, une méthodologie générale pour l'interprétation optimale d'une question arborescente en séquences de semi-jointures est présentée dans (CHIU80). Les résultats de (BERN79a) sont étendus en considérant l'opérateur de semi-jointure multi-attribut dans (BERN79b) et en considérant une classe de questions qui peuvent être solubles par des inequi-semi-jointures (BERN79c). Cette recherche conduit au développement de nouveaux algorithmes pour réaliser les semi-jointures de manière optimale.

Ce chapitre présente des algorithmes multiprocesseurs d'equi-semi-jointure et d'inequi-semi-jointure pour la machine bases de données SABRE. Paragraphes 2 et 3, le modèle d'architecture qui permet d'étudier ces algorithmes est à nouveau décrit avec les paramètres

nécessaires à l'évaluation des temps d'exécution de ces divers algorithmes. Ceci est partiellement redondant avec les paragraphes 2 et 3 du chapitre III et facilite la lecture de ce chapitre indépendamment du chapitre III. Paragraphe 4, deux algorithmes d'equi-semi-jointure sont présentés et comparés. Puis un algorithme pour réaliser l'inequi-semi-jointure est proposé au paragraphe 5. Ces algorithmes s'exécutent en un temps linéairement proportionnel au volume des relations opérandes et résultat et inversement proportionnel au nombre de processeurs traitants. L'usage de l'opérateur de semi-jointure est alors introduit pour améliorer les temps de réponse des jointures de relations. La méthode de jointure de deux relations est comparée, en termes de coût d'exécution, à la méthode de jointure des semi-jointures de chaque relation par l'autre. La méthode de jointure choisie pour les comparaisons est l'algorithme de jointure par produit, présenté au chapitre III, car il est simple et généralement le plus efficace avec un grand nombre de processeurs. Il est finalement montré que la méthode par semi-jointures est généralement supérieure.

2. MODELE D'ARCHITECTURE

L'environnement dans lequel les algorithmes de jointure sont étudiés est la machine bases de données SABRE (GARD81). Pour une description plus détaillée, le lecteur peut se reporter au chapitre I. Cette machine répond à des commandes qui lui sont transmises au moyen d'un protocole de manipulation de données. Ces commandes permettent de manipuler des relations à l'aide d'assertions basées sur la logique des prédicats. La machine est composée d'un ensemble de processeurs connectés au moyen d'un réseau d'interconnexion. Les processeurs échangent des commandes directement et les données au travers d'une mémoire cache. Ci-dessous,nous décrivons sous forme d'un modèle simplifié à trois niveaux les éléments essentiels de l'architecture qui

permettent de réaliser des jointures.

Le premier niveau de l'architecture est un ensemble de processeurs de filtrage de données (BANC80,FAUD80). Ces processeurs travaillent chacun en parallèle avec un flot de données provenant d'un disque. La stratégie de placement garantit que les relations puissent être lues en parallèle, grâce à un partitionnent des relations sur l'ensemble des unités de disques. Il a été montré dans (BORA81) que, si les lectures parallèles de disques étaient employées, l'approche processeur par disque donnait les meilleurs résultats, car elle évite la saturation du canal. Ces processeurs assurent la sélection "au vol" des données du flot répondant à une expression de sélection et le transfert des données retenues vers la mémoire cache. Une expression de sélection est composée d'une suite de références déterminant les attributs à projeter et d'une condition de sélection. Une condition de sélection est une conjonction (et) de disjonctions (ou) de clauses de la forme (Ai op C),où Ai est une référence d'attribut,op un des opérateurs de relation $<,\leq,=,\neq,\geq,>$ et C une constante. Les filtres qui seront développés (FAUD80) sont des processeurs relativement simples très proches de la forme générale des automates. Ils permettent de réaliser un traitement synchrone au flot, où en d'autres termes effectuent une selection à la vitesse de rotation des disques dans une page de mémoire secondaire correspondant généralement à une piste. Les filtres peuvent prendre en compte un nombre de prédicats qui ne dépend que des caractéristiques de la mémoire de l'automate, et non du flot de données et des expressions testées. Des évaluations (FAUD80) montrent que, pour une taille mémoire moyenne de 64 K octets, un filtre peut admettre jusqu'a 500 prédicats. Les filtres communiquent avec la mémoire cache au moyen d'un bus très rapide, de type multibus (INTE79).

Les données sélectionnées par les processeurs de filtrage sont

stockées dans la mémoire cache divisée en pages, qui constitue le deuxième niveau. Le gérant de la mémoire cache a pour role l'anticipation des lectures de pages et met en oeuvre un algorithme de remplacement des pages les moins récemment utilisées, pour les vider sur disque. Les filtres sont capables de travailler de la même manière sur des données stockées dans des pages en mémoire cache ou en mémoire secondaire. Ceci est utile pour opérer des sélections sur des données déjà en mémoire cache.

Le troisième niveau est un ensemble de processeurs spécialisés de jointure chargés d'exécuter les opérations de jointure et tri. sélections effectuées sur des relations. Chaque processeur de jointure dispose d'une mémoire propre de plusieurs pages. Il communique avec la mémoire cache pour acquérir ou restituer des pages lors de l'exécution d'une opération. Les processeurs de jointure sont connectés avec la mémoire cache par un réseau d'interconnexion spécifique. Pour obtenir des performances optimales, ce réseau de connexion doit avoir deux propriétés. La première est que plusieurs processeurs de jointure puissent lire ou écrire simultanément des pages en mémoire cache. La seconde est la possibilité de diffuser la même page à un nombre quelconque de processeurs de jointure en un seul transfert. Ce type de réseau est faisable (BRAT80, DEWI79), et différentes alternatives seront étudiées dans le projet SABRE. L'objet de cette thèse n'est pas d'y choisir une solution, car elle affectera les performances de tout le système. Pour notre propos, nous supposerons un réseau d'interconnexion ayant ces deux propriétés, n'introduisant pas de délai significatif.

3. PARAMETRES ET CRITERES D'ANALYSE

Afin de décrire différents algorithmes de jointure ou semi-jointure ,nous supposerons que nous avons à effectuer l'opération (jointure ou semi-jointure) sur deux relations R et S dans la relation résultat T. Les paramètres suivants seront utilisés pour l'évaluation des différents algorithmes proposés :

m : nombre de pages de R,

n : nombre de pages de S,

d : nombre de processeurs de filtrage associés à l'opération, le degré de parallélisme étant supposé le même pour les deux relations,

C : taille de la mémoire cache utilisable pour la jointure (en nombre de pages),

p : nombre de processeurs de jointure,

b+1 : taille de la mémoire propre d'un processeur de jointure (en nombre de pages) = b pages d'entrée + 1 page de résultat,

t : nombre moyen de tuples par page de chaque relation,

SJ : coefficient de sélectivité de la jointure défini par :

nombre de pages de T / (m*n) ,

SR : coefficient de sélectivité de la semi-jointure de R par S défini par :

nombre de pages résultant de la semi-jointure de R par S / m ,

SS : coefficient de sélectivité de la semi-jointure de S par R défini par :

nombre de pages résultant de la semi-jointure de S par R / n ,

j : nombre de valeurs distinctes d'attributs de jointure de R, S par page et nous noterons par mj et nj le nombre total de ces valeurs de R et S,

Nc : nombre de prédicats admis par un filtre,

Ct : coefficient d'occupation d'un attribut de semi-jointure dans un

tuple défini par : taille d'un attribut de jointure / taille d'un tuple.

Ceci permet de définir na, le nombre de pages obtenu apres projection de la relation S sur l'attribut de semi-jointure, na = n*Ct,

Ci-dessous, lors de la description détaillée des algorithmes, les processeurs sont supposés être numérotés de 0 à p-1, les pages de la relation R de 0 à m-1 et celles de S de 0 à n-1.

L'analyse des performances de nos algorithmes requiert une définition rigoureuse des paramètres de performances de base. Ces paramètres doivent mesurer le coût d'entrée-sortie, le coût de traitement et le coût de communication pour exécuter un algorithme dans un système à architecture multiprocesseur. Le temps d'exécution d'un algorithme à prendre en compte pour les comparaisons sera exprimé en termes de ces coûts de base. Les paramètres identifiés peuvent prendre différentes valeurs suivant les caractéristiques matérielles de l'architecture développée. L'hypothèse principale est que les données sont échangées et traitées par unités de pages. D'autre part, les relations traitées sont supposées suffisamment grandes pour ne pouvoir être entièrement contenues dans les mémoires locales des processeurs et (ou) en mémoire cache. Les paramètres de base à l'évaluation des performances sont maintenant détaillés.

3-1. Coût d'entrée-sortie :

Dans une architecture multiprocesseur où des processeurs échangent des données, la lecture ou l'écriture d'une page par un processeur se fait par l'intermédiaire de la mémoire cache et peut avoir deux valeurs, selon que la page est présente en mémoire cache ou réside sur une unité de mémoire secondaire. Le coût de transfert d'une page de mémoire cache vers un processeur ou entre deux processeurs est noté Tcp. Le coût de

transfert d'une page de mémoire secondaire vers la mémoire cache est noté Tdc et inclut le coût de la sélection qui s'effectue en parallèle a la lecture.

Il nous est alors utile de définir le coût moyen de lecture d'une page par un processeur en tenant compte de la probabilité que la page demandée soit déja en mémoire cache. Pour simplifier l'analyse, cette probabilité est assimilée au taux d'éfficacité de la mémoire cache, et notée F. Le rôle important du gérant de la mémoire cache est notamment le maintien de cette probabilité forte en anticipant les lectures de pages d'une même relation, ceci étant facilité par le fait que, généralement une relation référencée sera lue entièrement. Le coût moyen de lecture d'une page par un processeur est défini par CL :

$$CL = F*Tcp + (1-F)*(Tcp+Tdc)$$

De la même manière, nous pouvons évaluer le coût moyen d'écriture d'une page, en introduisant la probabilité qu'il y ait une page libre en mémoire, noté F'. Cette probabilité doit être maintenue faible pour une bonne occupation de la mémoire cache et relativement constante , en développant un algorithme de remplacement des pages les moins récemment utilisées. Le cout moyen d'écriture d'une page par un processeur, défini par CE, est :

$$CE = F'*Tcp + (1-F')*(Tcp+Tdc)$$

3-2. Cout de communication

Les transferts de pages sont vus comme des opérations d'entrée-sortie, aussi le coût de communication à prendre en compte se limitera aux messages de demande de pages et aux messages de réponse. Nous pouvons donc ajouter le coût de ces messages M aux coûts d'entrée-sortie définis précédemment et remplacer CL par CL + M(demande) + M(réponse) et CE par CE + M(demande) + M(réponse). Les méssages de

contrôle tels que ceux nécéssaires à l'allocation de processeurs sont en nombre négligeable par rapport aux messages de demande de page et réponse, et seront ignorés pour la clarté de l'analyse.

3-3. Cout de traitement

Un certain nombre de paramètres permet d'identifier les traitements éffectués par les processeurs lors de l'exécution parallèle d'un algorithme. Tout d'abord, V représente le coût d'une opération simple telle que comparer deux attributs ou calculer une fonction de hachage. Le coût pour transférer un tuple dans une zone mémoire est défini par I.

coût de jointure de deux pages : selon l'algorithme, les pages à joindre sont supposées non triées. Dans ce cas, deux manières de faire sont possibles. La première consiste à faire le tri interne des pages sur l'attribut de jointure puis à les fusionner. La seconde méthode compose le produit cartésien des tuples des pages deux à deux, ce qui est plus simple et plus performant que la première et n'exige que des opérations de comparaison plutôt que des transferts de tuples pour le tri interne. La jointure de deux pages de t tuples nécéssite t**2*V, où chaque tuple de la première est comparé à chaque tuple de la seconde. Le coût pour joindre deux pages non triées, défini par CJ, est :

CJ = t**2*V

4. ALGORITHMES D'EQUI-SEMI-JOINTURE

Si l'opérateur de comparaison Θ, dans la définition de la semi-jointure au paragraphe 1, est =, l'opération est appelée equi-semi-jointure. Dans ce paragraphe, nous proposons et évaluons deux algorithmes d'equi-semi-jointure, un employant des tableaux de bits et l'autre utilisant l'opérateur de sélection fourni par les filtres.

4-1. Equi-semi-jointure avec tableau de bits

La méthode proposée ici est une extension de la méthode monoprocesseur présentée dans (BABB79) pour effectuer les semi-jointures. En version de base, elle utilise un tableau de bits M pour aider l'opération. Dans notre version, ce tableau est dupliqué dans les mémoires propres des p processeurs de jointure. L'opération adaptée à un contexte multiprocesseur procède essentiellement en deux étapes. D'abord, les valeurs d'attributs sources de semi-jointure sont lues en parallèle par d processeurs de filtrage et transférées vers la mémoire cache, pour être enfin compactées dans des pages. La relation résultant de la projection sur l'attribut de semi-jointure de la relation S est nommée S' et comporte na pages. Ces pages contenant des attributs sources sont alors réparties entre p processeurs de jointure. Chaque processeur lit ainsi na/p pages de la relation source S' et écrit chaque valeur d'attribut source en ordre ascendant dans une zone W de mémoire locale sauf si cette valeur y est déja présente. Simultanément, pour chaque valeur v d'attribut source lue, un bit du tableau M est marqué. L'adresse du bit à marquer est déterminée par l'application d'une fonction de hachage h, où plus précisément M(h(v)) est mis a 1. A la fin de cette premiere étape, le tableau M de chaque processeur reflète une partie des valeurs d'attributs de la relation source. L'union logique de ces p tableaux en un seul représentera toutes les valeurs d'attributs de la relation source. Pour ce faire, les tableaux sont transmis à un processeur de jointure qui réalise leur union et diffuse en résultat le tableau M uni aux p-1 autres processeurs. Il reste encore a fusionner les listes intermédiaires W de chaque processeur en une seule, qui contiendra toutes les valeurs distinctes d'attributs de semi-jointure. Ceci peut être réalisé en une étape de fusion d'ordre b par un processeur de jointure si $p \leq b$ ou en $\log_b p$ étapes si $p > b$, où le nombre de processeurs utilisés est divisé par b à chaque

étape. La liste W fusionnée , ne contenant pas de doubles, est diffusée aux p processeurs. Cet algorithme n'est applicable que si la mémoire propre d'un processeur peut contenir le tableau de bits et la liste W. Ceci sera vrai dans la plupart des cas. Si la liste de valeurs d'attributs sources était très grande, le problème se ramenerait à celui de la jointure étudié au chapitre III. La deuxième étape consiste à répartir, avec sélection préalable par les filtres, la relation cible R sur les p processeurs. Chaque processeur reçoit une page R et, chaque tuple R dont la valeur v d'attribut de jointure vérifie B(h(v))=1, est projeté dans la relation résultat si v appartient à W. Cette dernière vérification est nécéssaire pour éviter les collisions possibles dues au hachage (rappelons que plus d'un tableau de bits peut être utilisé pour diminuer la probabilité de collisions.

Figure 2, l'algorithme exécuté par le processeur de jointure k avec q fonctions de hachage et tableaux de bits est présenté sous forme d'une procedure SJOINDREH (R,S,T,A,B), qui réalise la semi-jointure de la relation R par la relation S sur les attributs A et B, avec écriture dans la relation résultat T. BL et BE désignent les tampons de lecture et d'écriture d'une page du processeur. Sup(a) est une fonction qui désigne l'entier immédiatement supérieur à a. LIRE-PAGE (S,BL) est une commande de lecture séquentielle d'une page de S dans le tampon BL envoyée à la mémoire cache. ECRIRE-PAGE (BE,T) commande l'écriture séquentielle du tampon BE dans la relation T. Il est nécéssaire pour cela que le gérant de la mémoire cache dispose d'un pointeur courant de lecture ou écriture pour une relation. La procedure INSERER (B,W) insère l'attribut B selon l'ordre ascendant dans la liste W, s'il n'y est pas déjà. UNION (Mi) donne le résultat de l'union des p tableaux de bits Mi et FUSION (W) le résultat de la fusion des p listes W dans W. AJOUTER (r,BE) permet d'ajouter le tuple r dans le tampon BE.

```
procedure SJOINDREH (R,S,T:relation; A,B:attribut);

1. pour i=1 à Sup(na/p) faire
       LIRE-PAGE (S',BL);
       pour chaque attribut B dans BL faire
           Ml(hl(B))=1; ... Mq(hq(B))=1;
           INSERER (B,W);

UNION (Ml, ...Mq);
FUSION (W);

2. pour i=1 à Sup(m/p) faire
       LIRE-PAGE (R,BL);
       pour chaque tuple r de BL faire
          si Ml(hl(r.A))=1 et ... Mq(hq(r.A))=1 alors
            si "r.A dans W" alors
               AJOUTER (r,BE);
               si "BE plein"  alors ECRIRE-PAGE (BE,T);
fin-SJOINDREH
```

figure 2 : algorithme de semi-jointure avec tableaux de bits

4-1-1. Evaluation de l'algorithme avec tableaux de bits

Le coût total d'exécution de l'algorithme est la somme du coût d'entrée-sortie (qui inclut le coût de communication) et du coût de traitement CPU, ce qui peut être représenté par :

$$C(SJOINH) = C_{ES}(SJOINH) + C_{CPU}(SJOINH)$$

1) Coût d'entrées-sorties

Les transferts de pages inter-processeurs sont considérés comme des opérations d'entrées-sorties. Aussi nous prenons en compte le coût de projection de la relation S dans la relation S', le coût de lecture de la relation S' lors de l'étape 1, le coût de transferts nécessaires aux opérations d'union des tableaux de bits et de fusion des listes, le coût de lecture de la relation R lors de l'étape 2 et le coût d'écriture de la relation résultat.

La relation S est projetée sur les attributs de semi-jointure dans la relation S'. Cette opération nécessite de lire S entierement par d processeurs de filtrage, ce qui coûte (n/d)*Tdc. La relation S' comportant na pages est lue en parallèle par p processeurs, soit un coût

de (na/p)/CL. L'opération d'union des p tableaux, en supposant qu'ils tiennent dans une page, nécessite p transferts pour leur lecture par un des processeurs qui fait l'opération et diffuse le résultat en un transfert, soit un coût de (p+1)*Tcp. La fusion des p listes d'attributs est réalisée en $\log_b p$ étapes de fusion d'ordre b, et en supposant la taille moyenne d'une liste de b/2 pages, le coût de transfert est $\log_b p*(b/2)*Tcp$. La deuxième étape consiste à lire la relation R de m pages pour un coût de (m/p)*CL et à écrire la relation résultat, dont la taille dépend du coefficient de sélectivité de semi-jointure SR, soit un coût de (m/p)*SR*CE.

Finalement, le coût en entrées-sorties de l'algorithme par hachage est :

$$C_{ES}(H) = (n/d)*Tdc + (na/p)*CL + ((p+1)+(\log_b p*(b/2)))*Tcp + (m/p)*(CL+(SR*CE))$$

2) Cout CPU

Nous considérons maintenant le coût de traitement CPU de l'algorithme par hachage. Lors de l'étape 1, pour chaque page de la relation S' lue, qui contient t/Ct attributs, une fonction de hachage est appliquée à chaque attribut, qui est ensuite inséré en ordre dans la liste W. Nous notons le coût d'insertion d'un attribut dans W par Iw. Le cout de traitement CPU de cette étape est alors :

(na/p)*(t/Ct)*(V+Iw)

L'opération d'union des p tableaux par un processeur nécessite une comparaison de chacun d'eux, et en considérant un tableau comme un certain attribut, le coût de comparaison est p*V. La fusion des listes W de chaque processeur a un cout noté Cf.

Lors de la seconde étape, pour chaque page de la relation R lue, une fonction de hachage est appliquée aux t attributs et pour les tuples retenus au nombre de t*SR par page, un accès à la liste W est encore

nécéssaire pour s'assurer qu'il n'y a pas de collisions. La liste W étant triée, et W comportant nj valeurs d'attributs, une recherche dichotomique peut être appliquée (KNUT73, p.406) pour chaque tuple, avec un coût de $\log_2 nj*V$ par tuple. Le coût de transfert des tuples dans la page résultat est SR*I par page lue de la relation R. Le coût de la deuxième étape est alors : $(m/p)*((t+ (t*SR*\log_2 nj))*V + SR*I)$.

Le coût de traitement CPU de l'algorithme est alors :

$$C_{CPU}(H) = (na/p)*(t/Ct)*(V+Iw) + p*V + Cf + (m/p)*((t +(t*SR*\log_2 nj))*V +SR*I)$$

4-2. Equi-semi-jointure par sélection

La méthode décrite ici profite pleinement du mécanisme de sélection évolué offert par les processeurs de filtrage (FAUD80) pour traiter l'equi-semi-jointure. L'opération comporte deux étapes. La première est similaire à celle de l'algorithme précédent à la différence près qu'elle n'utilise pas de tableaux de bits. Les valeurs d'attributs sources de semi-jointure sont lues en parallèle, compactées dans des pages en mémoire cache et réparties entre p processeurs de jointure. Chaque processeur écrit les valeurs d'attributs sources dans son tampon de lecture triées dans une zone W, en éliminant les valeurs en double et répète l'opération jusqu'à la fin de lecture de la relation source. Les listes W sont alors fusionnées de la même facon que précédemment.

La deuxième étape réalise la semi-jointure en restreignant la relation cible aux tuples dont la valeur d'attribut de semi-jointure est présente dans la liste W. A partir de la liste W, un processeur de jointure arbitrairement choisi construit des conditions de sélection dans la relation cible en un constituant un prédicat de restriction comme une conjonction de clauses de la forme (A=Bi), où A désigne l'attribut de semi-jointure de R et les Bi sont des valeurs distinctes dans la liste W. Plusieurs opérations de sélection sont nécéssaires si

le nombre de valeurs Bi dans la liste W qui vaut nj est plus grand que Nc, le nombre maximum de clauses admises par un filtre. Le nombre d'opérations de sélection est Sup (nj/Nc). Comme chaque opération de sélection exige la lecture de la relation R entière, la performance de cet algorithme dépend de la puissance de traitement du filtre. Rappelons qu'un filtre relativement simple, possédant une mémoire moyenne, peut accepter un nombre élevé de prédicats.

Figure 3, l'algorithme exécuté par le processeur de jointure k est présenté sous forme d'une procedure SJOINDRES avec les mêmes paramètres que ceux de la procedure SJOINDREH décrite en 4-1. La deuxième étape de l'algorithme est exécutée par ce processeur k et est donc décrite dans la même procedure. La première étape invoque les memes procedures que SJOINDREH. SELECTER (R,T,LAT,COND) est une commande destinée aux filtres de selectionner dans la relation R les attributs définis par LAT (projection) répondant au prédicat COND (restriction) et de les écrire dans la relation T en mémoire cache. Enfin Min(a,b) désigne la valeur minimum de a et b. COND et W sont manipulés comme des ensembles.

```
procedure SJOINDRES (R,S,T:relation; A,B:attribut);

1. pour i=1 à (na/p) faire
      LIRE-PAGE (S,BL);
      pour chaque attribut B de BL faire INSERER (B,W);

FUSION (W);

2. pour i=1 à Sup(nj/Nc) faire
     j=(i-1)*Nc;
     COND=(A=W(j))
     pour l=j+1 à Min(j+Nc-1,nj) faire
         COND =COND ou (A=W(l));
     SELECTER (R,T,LAT,COND);
fin-SJOINDRES
```

figure 3 : algorithme de semi-jointure avec sélection

4-2-1. Evaluation de l'algorithme par sélection

De la même façon que pour l'algorithme précédent, le coût d'exécution est égal à la somme des coûts d'entrée-sortie et de traitement CPU.

1) Cout d'entrées-sorties

L'algorithme comporte deux étapes dont la première est la lecture en parallèle des attributs sources par les processeurs de jointure et la deuxième la selection par les filtres. La première étape , similaire à celle de l'algorithme précédent, est la projection de la relation S dans S' par d processeurs de filtrage et la lecture de S' en parallèle par p processeurs. Cette étape coûte (n/d)*Tdc + (na/p)*CL.

L'opération de fusion est identique à celle de l'algorithme précédent, soit un coût de $(\log_b p*(b/2))$ *Tcp.

La deuxième étape est la selection par d filtres en parallèle dans la relation R de m pages à la vitesse du flot. Une sélection sur la relation entière coûte (m/d)*Tdc et peut être répétée nj/Nc fois si nj est plus grand que Nc. Il est à noter que la sélection inclut le coût d'écriture de la relation résultat en mémoire cache, car la condition de filtrage synchrone au flot nécessite des pages disponibles en mémoire cache durant l'opération afin de ne pas être ralenti. Le coût de la deuxième étape est donc : (nj/Nc)*(m/d)*Tdc. Finalement, le coût en entrées-sorties de l'algorithme est :

$$C_{ES}(SJ) = (n/d)*Tdc + (na/p)*CL + (\log_b p*(b/2))*Tcp + (nj/Nc)*((m/d)*Tdc)$$

2) Coût CPU

Le coût CPU est donné par les insertions dans la liste W contenant les valeurs d'attributs de semi-jointure sources et est le même que celui de l'algorithme précédent : (na/p)*(t/Ct)*Iw.

Le coût de fusion de p listes est noté Cm.

L'initialisation des opérations de sélection avec nj attributs lors de la seconde étape est nj*V. Le coût CPU de l'algorithme est alors :

$$C_{CPU}(SJ) = (na/p)*(t/Ct)*Iw + Cm + (nj*V)$$

4-3. Comparaisons

A partir des formules développées aux paragraphes précédents, nous pouvons comparer les performances de ces deux algorithmes d'equi-semi-jointure. Les résultats sont présentés figure 4 et 5. Les hypothèses concernant les possibilités des processeurs, paramétrés par les coûts de base définis au paragraphe 2 sont décrites en annexe 1.

La figure 4 illustre le comportement des algorithmes en fonction de la taille des relations, supposées composées du meme nombre de pages (m=n). La configuration choisie comme représentative d'une architecture multiprocesseur simple comporte 8 filtres et 8 processeurs de jointure, dotés d'une mémoire propre de 5 pages (b=4). Un filtre est supposé admettre un nombre de prédicats Nc égal a 100. La relation S', résultant de la projection de la relation S sur l'attribut de jointure comporte na=0.1*n pages. L'étape qui consiste à lire la relation S' étant la même pour les deux algorithmes, la valeur de na n'influe donc pas les comparaisons, tout comme le coût Cf de fusion des listes W. La figure 4 représente 4 courbes. Les deux courbes de coût de l'algorithme avec tableaux de bits (H) varient linéairement en fonction de n et different de par le coefficient de sélectivité, qui n'intervient en fait que très faiblement sur le coût de constitution de la relation résultat. Les deux courbes de coût de l'algorithme par sélection (JS) different par le nombre nj de valeurs distinctes d'attribut de semi-jointure et mettent en évidence la forte dépendance de la performance de l'algorithme JS du rapport Sup(nj/Nc), qui représente le nombre de passes nécéssaires à la sélection. Dès que nj/Nc devient supérieur à

l'entier i, il faut i+1 passes pour faire la sélection, ceci étant montré par la forme en escalier de ces courbes. L'algorithme est plus performant lorsqu'il y a peu de valeurs distinctes d'attributs de semi-jointures, c'est à dire beaucoup de synonymes.

Plus généralement, l'algorithme par sélection est meilleur que l'algorithme H, si l'équation approximée :

$$(m/p)*(CL+(SR*CE)) - (Sup(nj/Nc) *((m/d)*Tdc)$$

est strictement positive.

La figure 5 décrit les coûts d'exécution de ces algorithmes en fonction du nombre de processeurs, tel que p=d. Les performances de ces algorithmes sont proportionnelles au nombre de processeurs , et la différence entre les courbes JS (nj=0.2*n) et JS (nj=0.4n) est encore due au rapport (nj/Nc). Lorsque nj=0.2*n, l'algorithme par selection et l'algorithme avec tableaux de bits ont le même coût. Au dela de nj=0.2*n, l'algorithme par sélection est moins performant. La performance de l'algorithme par sélection dépend de la possibilité du filtre à admettre un grand nombre de prédicats. Ceci est un objectif des concepteurs de filtres orientés notemment vers le traitement de texte.

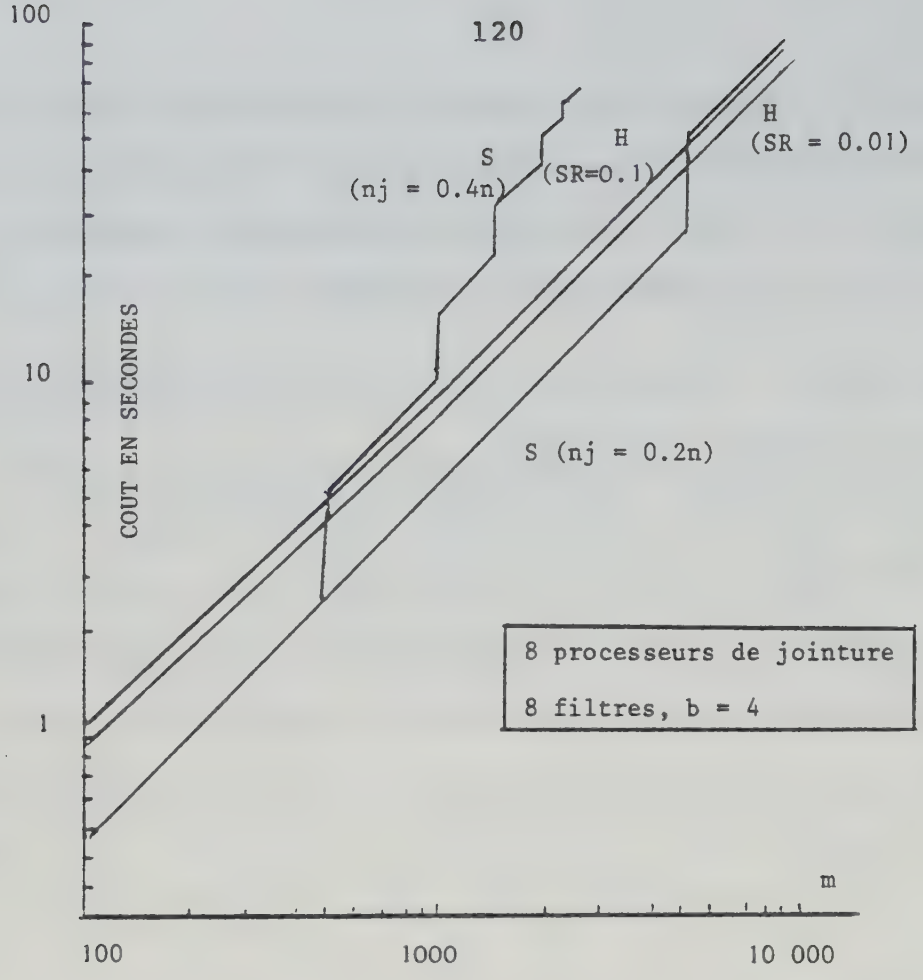


figure 4 : Coûts d'exécution des algorithmes en fonction de la taille des relations

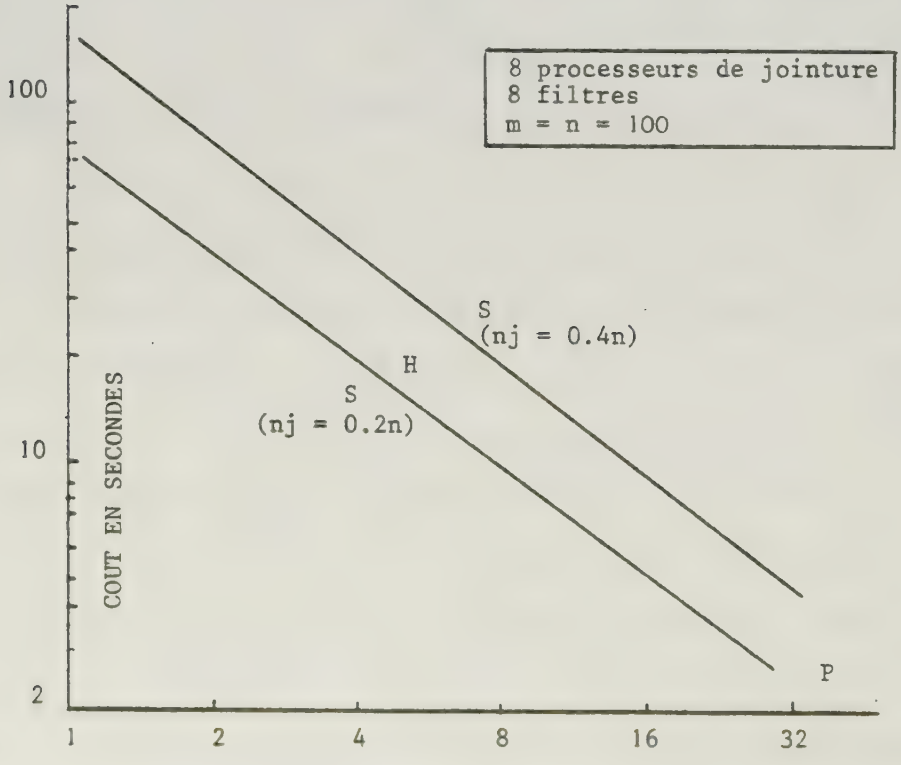


figure 5 : Coûts d'exécution des algorithmes en fonction du nombre de processeurs

5. ALGORITHMES D'INEQUI-SEMI-JOINTURES

L'inequi-semi-jointure correspond au cas où l'opérateur de comparaison est parmi $<,\leq,\geq,>$. Dans ce paragraphe, nous proposons un algorithme multiprocesseur très simple pour réaliser l'inequi-semi-jointure dans une machine bases de données. La méthode repose sur l'observation suivante : l'inequi-semi-jointure consiste à retenir tous les tuples r de la relation cible tels que l'attribut de semi-jointure A vérifie "r.A op X", où X est soit la valeur minimum d'attribut de semi-jointure de la relation source si op est $>$ ou $\geq$, soit la valeur maximum si op est $<$ ou $\leq$. La recherche d'une valeur minimum ou maximum dans un ensemble de données à addressage associatif est une opération faisable par les processeurs de filtrage décrits dans (FAUD80). L'algorithme comporte alors deux étapes. Tout d'abord, les valeurs distinctes d'attributs de jointure sont lues en parallèle par les processeurs de filtrage. Chaque valeur est comparée à la valeur minimum ou maximum courante maintenue dans la mémoire propre du filtre et la remplace par la valeur lue si nécessaire. Ainsi d processeurs obtiennent des résultats partiels qui sont intégrés en un seul pour avoir la valeur de X, ceci pouvant être réalisé par un processeur. L'inequi-semi-jointure est enfin réalisée en sélectionnant en parallèle par les d filtres les tuples de la relation cible qui vérifient A op X. Cet algorithme a l'avantage de ne pas utiliser la mémoire cache pour la lecture d'opérandes et ne nécessite qu'une lecture des relations.

Figure 6, l'algorithme exécuté par un processeur de jointure qui initie l'opération est décrit sous forme de la procedure ISJOINDRE avec les mêmes paramètres que les procedures d'equi-semi-jointure, plus l'opérateur op. Les fonctions MIN et MAX (relation, attribut) sont des commandes au processeur controlant les filtres, de rechercher la valeur minimum ou maximum de la relation source. La commande de sélection est

identique à celle de la figure 3.

```
procedure ISJOINDRE (R,S,T:relation; A,B:attribut; op:opérateur);

     si "op est ≥ ou > "
         alors  X= MIN(S,B)
         sinon  X= MAX(S,B);
     SELECTER (R,T,"A op X");
fin-ISJOINDRE
```

figure 6 : algorithme d'inequi-semi-jointure

5-1. Evaluation de l'algorithme

Le coût d'exécution CPU de la procedure ISJOINDRE est négligeable (4*V). Le coût en entrées-sorties est le coût de lecture de la relation source de n pages pour calculer le minimum ou maximum en parallèle par d filtres, plus le coût de lecture de la relation cible de m pages pour appliquer la sélection. La relation résultat est écrite directement en mémoire cache et le coût d'écriture est inclu dans le coût de la sélection. Le coût d'exécution de cet algorithme est alors :

$$C(IJ) = ((n/d) + (m/d)) * T_{dc}$$

Il est à remarquer que pour ne pas ralentir les processeurs de filtrage lors de l'exécution parallele de la sélection, le gérant de la mémoire cache doit maintenir des pages libres comme tampons résultats.

6. JOINTURES UTILISANT DES SEMI-JOINTURES

Les algorithmes de semi-jointure proposés ci-dessus s'exécutent en un temps d'exécution linéairement proportionnel au volume des opérandes. La qualité de l'opérateur de semi-jointure étant de réduire le volume d'une relation aux seuls tuples participant a la jointure, nous pensons qu'il est intéressant dans certains cas de remplacer la jointure de deux relations par deux semi-jointures suivies de leur jointure. Afin de comparer les deux alternatives (jointures ou semi-jointures), l'algorithme de jointure multiprocesseur par produit est choisi car il

est à la fois simple et généralement performant avec un haut degré de parallélisme.

6-1. L'algorithme de jointure par produit

Nous rappelons brievement l'algorithme présenté au chapitre III. Soient à joindre deux relations R et S, la plus petite relation choisie comme externe est répartie séquentiellement sur p processeurs de jointure puis l'autre relation, interne, est diffusée page par page aux processeurs. Chaque processeur joint alors chaque page de la relation externe qu'il recoit à toutes les pages de la relation interne. Si la relation externe ne peut être contenue par les p processeurs, le même procédé doit être répété pour les pages suivantes de cette relation.

Si la relation externe (soit R) peut être mémorisée par les p processeurs, c'est à dire m=p*(b-1), le temps d'exécution de cet algorithme est :

C (PC) = C(lire (b-1) pages de R) + n*C(diffuser une page de S)
+ n*C(joindre (b-1) pages de R avec une page de S)

Si $m \geq p*(b-1)$, le même procédé doit être répété $m/(p*(b-1))$ fois. Le coût de jointure de (b-1) pages de R avec une page de S est (b-1)*CJ. Le coût d'écriture du résultat de la jointure de deux pages, dépendant du coefficient de sélectivité de la jointure défini par SJ est : SJ*((t*I)+CE). Finalement, le temps d'exécution de l'algorithme par produit est :

C (PC) = m/(p*(b-1)) *(CL + N*(CL +(b-1)*(CJ +SJ*((t*I)+CE)))

6-2. Comparaisons

La jointure par produit cartésien des relations, notée PC, est comparée à la jointure par produit des semi-jointures d'une relation par une autre, notée PCSJ. L'algorithme de semi-jointure avec tableaux de

bits est choisi pour les comparaisons car il est plus général que l'algorithme par sélection et plus fréquemment invoqué que l'algorithme d'inequi-semi-jointure. La condition d'application de cet algorithme est que la mémoire propre de chaque processeur soit suffisamment grande pour contenir la liste de valeurs distinctes d'attributs de semi-jointure, plus deux pages (entrée, sortie), plus une page pour les tableaux de bits. Cette condition n'est pas vérifiée lorsque cette liste est grande, si par exemple la relation source est grande avec peu d'attributs de jointure synonymes. Le problème se ramène alors au problème de la jointure et en ce cas, l'algorithme de jointure par hachage , présenté au chapitre III, serait probablement élu, car une des relations opérandes serait précédée par une projection et plus petite que l'autre. La comparaison de ces deux méthodes (jointures ou semi-jointures) sont illustrées par les figures 7, 8 et 9, décrivant l'évolution de leur temps d'exécution en fonction de paramètres déterminant leurs performances. Les hypothèses concernant les caractéristiques de la configuration sont les mêmes qu'en 4-3.

La figure 7 décrit les temps d'exécution des deux méthodes en fonction du rapport SJ' égal à : taille de la jointure de R et S / (m'*n'), où m' et n' sont respectivement les tailles de semi-jointures de R par S et S par R. Pour chaque méthode, trois coefficients de sélectivité de jointure SJ sont considérés (0.5, 0.16 et 0.016) et les tailles des relations sont importantes (m=n=1024 pages). La variation de SJ' influe de manière significative sur l'algorithme PCSJ : quand SJ' tend vers 0, la taille de la jointure tend vers la taille de la plus grande semi-jointure et quand SJ' tend vers 1, la taille de la jointure tend vers la taille du produit cartésien des semi-jointures et dans ce cas, la semi-jointure diminue beaucoup la taille de la relation initiale. La méthode PCSJ est généralement plus efficace que la méthode PC et devient beaucoup plus performante à mesure que le coefficient de

sélectivité de la jointure devient faible. La méthode PC n'est efficace que lorsque la sélectivité de la jointure est forte avec un rapport SJ' faible, c'est à dire lorque la jointure tend vers le produit cartésien des relations.

La figure 8 illustre le comportement des méthodes PC et PCSJ en fonction de la taille des relations supposées égales, pour le coefficient de sélectivité de jointure 0.016 et le rapport SJ' =0.4. La supériorité de la méthode PCSJ est évidente puisque le coût des semi-jointures est négligeable par rapport au coût de jointure et que les relations opérandes y sont plus petites que pour la méthode PC.

L'augmentation du nombre de processeurs pour deux jointures différentes, figure 9, démontre encore la performance de la méthode PCSJ, pour un coefficient de sélectivité de jointure moyen (0.016). L'écart entre les méthodes diminuerait quelque peu en augmentant ce coefficient.

Ces comparaisons montrent que la méthode consistant à précéder l'application de l'algorithme multiprocesseur de jointure par les semi-jointures des relations opérandes diminue généralement beaucoup le coût de la jointure, ce qui confirme l'intérêt à développer cet opérateur.

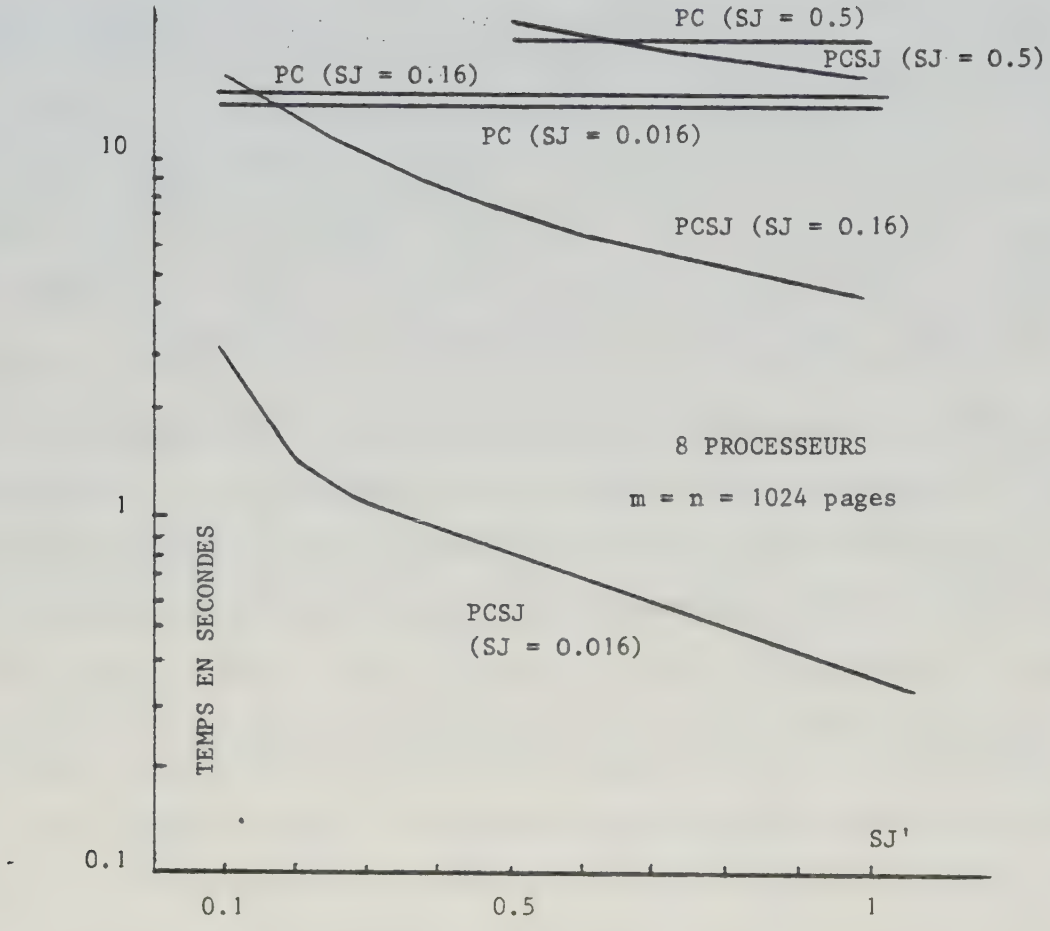


figure 7 : Comparaison des méthodes de jointure par produit (PC) et par semi-jointures (PCSJ) en fonction de SJ'

SJ' = taille de la jointure de R et S / (m'*n')
ou
m' = taille de la semi-jointure de R par S
n' = taille de la semi-jointure de S par R

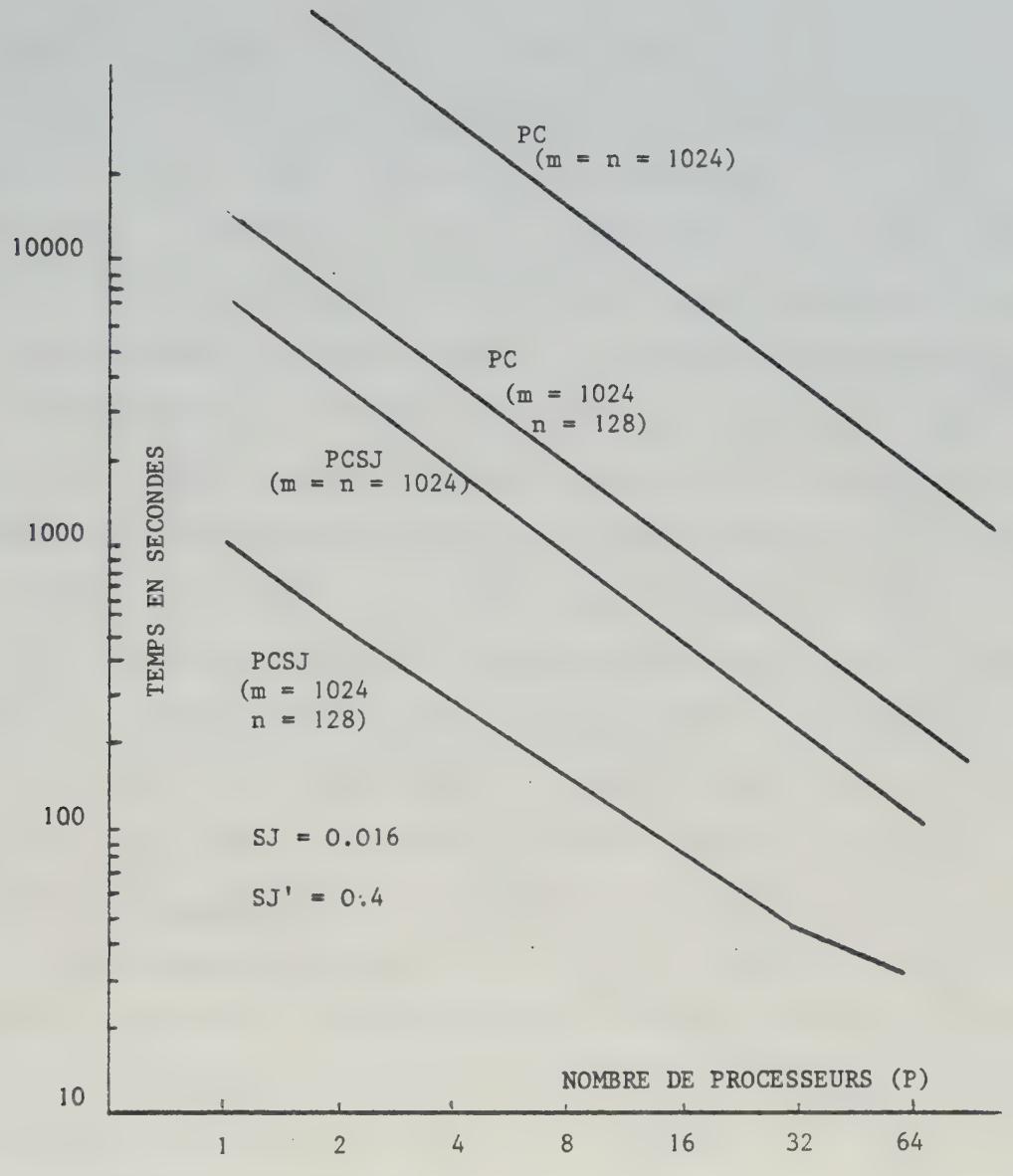


figure 8 : Comparaisons des algorithmes PC et PCSJ en fonction des tailles de relations

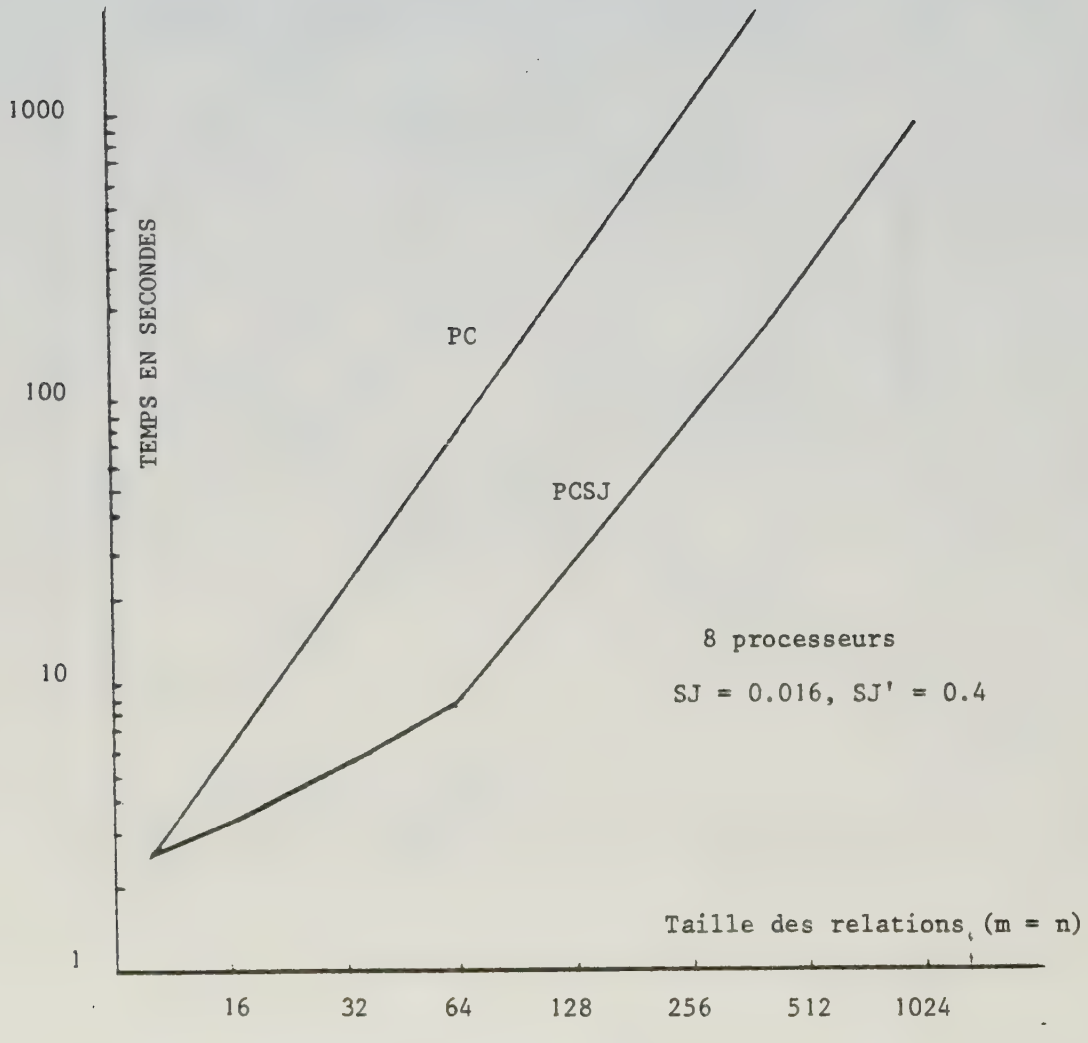


figure 9 : Comparaisons des algorithmes PC et PCSJ en fonction du nombre de processeurs

7. CONCLUSION

Dans ce chapitre, il a premièrement été rappelé l'intérêt de l'opérateur de semi-jointure dans un environnement de base de données réparties afin de réduire le coût de transmission lors de l'exécution d'opérations binaires. Puis s'intéressant au coût de transfert inter-processeurs au sein d'une machine bases de données multiprocesseur nous avons présenté et analysé des algorithmes parallèles pour réaliser les semi-jointures. Les éléments d'architecture de SABRE utiles pour réaliser des semi-jointures ont été rappelés. Un modèle décrivant les paramètres de performances de base incluant les coûts d'entrées-sorties, de traitement CPU et de transmission de messages a permis l'évaluation des algorithmes proposés. Deux algorithmes d'equi-semi-jointure ont été analysés. L'algorithme utilisant des tableaux de bits donne de bons résultats; son coût d'exécution est proportionnel à na/p + m/p + r/p, où na est le nombre de pages contenant les attributs sources de semi-jointure, m le nombre de pages de la relation cible, r le nombre de pages de la relation résultat et p le nombre de processeurs. La principale condition d'application de cet algorithme est que la liste contenant les valeurs distinctes d'attributs de semi-jointure puisse être contenue en mémoire locale d'un processeur.

Un second algorithme d'equi-semi-jointure utilisant la sélection à été proposé. Il repose essentiellement sur la puissance de traitement des processeurs de filtrage associés aux unités de mémoire secondaire. Il est supérieur au premier algorithme quand l'opération peut être effectuée en une seule passe, c'est à dire quand le nombre de valeurs distinctes d'attributs de semi-jointure est inférieur ou égal au nombre de prédicats que peut admettre le filtre. Sinon, l'algorithme avec tableaux de bits est meilleur.

Un algorithme d'inequi-semi-jointure tres simple a été proposé.Il utilise les fonctions maximum et minimum offertes par les filtres et ne se sert de la mémoire cache que pour l'écriture de la relation résultat. Cet algorithme a un cout d'exécution de l'ordre de m/d + n/d, où m et n sont les tailles des relations opérandes et d le nombre de filtres associés a l'opération.

Des comparaisons ont alors été faites entre deux méthodes de jointure : la jointure des relations opérandes et la jointure des semi-jointures de l'une par l'autre. L'algorithme de jointure par produit, présenté au chapitre III, a été choisi pour les comparaisons. Il a été finalement montré que précéder l'opération de jointure par des semi-jointures donnait généralement les meilleurs résultats. Cette méthode est très efficace lorsque les relations opérandes sont grandes et que la taille de la jointure est petite. Ceci confirme la nécessité d'implémenter l'opérateur de semi-jointure dans une telle machine bases de données.

CHAPITRE V

SPECIFICATIONS D'UN PROCESSEUR DE JOINTURE ET TRI

1. INTRODUCTION

L'architecture de la machine bases de données SABRE a été présentée au chapitre I comme contexte de notre étude. L'architecture opérationnelle de la machine est composée de processeurs communiquant au moyen d'un réseau d'interconnexion et d'une grande mémoire cache. Les processeurs sont spécialisés et assignés à une ou plusieurs fonctions. De plus, un ensemble de processeurs peut se partager la charge due à une même fonction, généralement coûteuse en temps d'exécution. Ceci introduit deux types de parallélisme : inter-requête et intra-requête. Celui qui nous intéresse ici est le parallélisme intra-requête, appliqué aux opérations qui exigent un traitement répétitif sur des relations trop grandes pour être contenues en mémoire. Afin de supporter ces opérations de base du modèle relationnel, des processeurs spécialisés réalisent les opérations de jointure, tri, semi-jointure et élimination des doubles.

Une partie importante de notre étude a été consacrée à la présentation et l'étude d'algorithmes multiprocesseurs pour réaliser efficacement les jointures et les tris, au chapitre III, et les semi-jointures au chapitre IV. L'opération d'élimination des doubles, qui suit une opération de projection, telle que la traitent les filtres, est similaire au tri.

Ce chapitre décrit les spécifications d'une classe de processeurs de jointure et tri (PJTs) assignés à ces quatre fonctions de base. Cette classe de processeurs intervient avec les processeurs de sélection (filtres) et le processeur gérant la mémoire cache, au niveau fonctionnel le plus interne de l'architecture, qui manipule des partitions de mémoire secondaire (voir chapitre I, 4-1). L'organisation de ce chapitre est la suivante. Tout d'abord, les fonctions supportées par les PJTs et les algorithmes mis en oeuvre sont rappelés. Les

services rendus par ces processeurs décrivent l'interface externe, par une liste de primitives. Les primitives sont exprimées en langage PASCAL, qui permettra la validation des interfaces entre les différentes classes de processeurs, pour conduire à la mise au point d'une première version simulée de SABRE. La stratégie d'allocation des PJTs à une opération parallèle est développée par un noyau d'exécution répartie (NER) décrit dans (BERN81), qui a pour rôle plus général le contrôle et la synchronisation de processus exécutés sur différents processeurs. L'interface interne des PJTs décrit l'ensemble des primitives exécutées en parallèle, et appelées par une primitive de l'interface externe. Finalement, les services demandés par les PJTs sont définis. Ils concernent l'obtention de pages opérandes et la restitution de pages résultats en mémoire cache. Ces services sont décrits par une liste de primitives, inclues dans l'interface externe du gérant de la mémoire cache. Les jeux de tests des primitives PASCAL du PJT, qui sera simulé pour la version SABRE monoprocesseur sont donnés en Annexe 2.

2. FONCTIONS DES PROCESSEURS DE JOINTURE ET TRI

Les fonctions de ces processeurs sont d'une part les opérateurs relationnels binaires de jointure et semi-jointure, et d'autre part les opérateurs coûteux que sont le tri et l'élimination des doubles après projection. Le choix d'assigner une même classe de processeurs, comme les PJTs, à plusieurs fonctions se justifie par la volonté d'éviter une trop grande spécialisation, qui pourrait conduire à surcharger certains processeurs, car leur fonction est très demandée, alors que d'autres, assignés à une autre fonction resteraient inactifs. Les opérations assignées aux PJTs sont générées lors de la décomposition et optimisation de question par le processeur de manipulation de base. Ce processeur accède le shéma de la base et fournit les paramètres nécessaires à l'appel d'une primitive des PJTs. Les opérandes sont des

relations permanentes ou temporaires, qui seront toujours obtenues en mémoire cache. L'avantage important est que les relations intermédiaires d'une question n'auront pas à être placées sur mémoire secondaire.

Les PJTs communiquent avec le gérant de la mémoire cache directement au moyen d'un réseau d'interconnexion. Les filtres, qui assurent la sélection des données dans des partitions associatives, transfèrent les données compactées dans des pages vers la mémoire cache. L'unité d'enregistrement physique de SABRE est la partition et son choix est justifié par la méthode de placement par regroupement multi-attribut (KARL80). Compte tenu de la technologie actuelle, la partition est assimilée à une piste de disque. Les PJTs et la mémoire cache manipulent les données dans des pages. La page est l'unité d'enregistrement logique, et est plus fine que la partition. La distinction page-partition permet une certaine indépendance de la machine aux mémoires secondaires. En effet, le choix de la taille de la partition est lié au support physique, car la partition est l'unité de balayage par les filtres, alors que le choix de la taille de la page est lié au débit du réseau d'interconnexion ainsi qu'au degré de parallélisme à atteindre (une taille de page trop grande pourrait conduire à diminuer le nombre de processeurs à mettre en oeuvre).

Les PJTs sont affectés à quatre fonction importantes. Ces fonctions sont généralement coûteuses en temps d'exécution. Elles seront donc exécutées par un ensemble de processeurs travaillant en parallèle, avec pour objectif de diminuer le temps d'exécution par le nombre de processeurs. Un problème important à résoudre par le novau d'exécution est d'optimiser l'allocation des processeurs, en favorisant le parallélisme inter et intra-requête. Dans la mesure du possible, le nombre maximum de processeurs est alloué à une opération, et les autres

processeurs restés disponibles peuvent être alloués à une autre opération. Nous rappelons maintenant les algorithmes qui seront implantés sur les PJTs.

2-1. Jointure

La première fonction des PJTs est la jointure de deux relations permanentes ou temporaires dans une relation cible temporaire, en appliquant éventuellement une projection sur des attributs spécifiés. Cette projection est souvent nécessaire lorsque les attributs de jointure ne sont pas demandés dans le résultat. La relation cible est créée en mémoire cache, afin d'être disponible pour d'éventuelles autres opérations. Au chapitre III, trois algorithmes multiprocesseurs de jointure ont été décrits et analysés : l'algorithme par produit, l'algorithme par tri et l'algorithme par hachage. L'algorithme par produit se prête aisément au traitement de l'inequi-jointure, car il compare tout tuple d'une relation à tout tuple d'une autre. L'algorithme par tri produit une relation cible triée sur l'attribut de jointure. Dans ce chapitre III, il est montré que chacun des algorithmes possède un créneau d'application défini par les caractéristiques des relations à joindre et de la configuration (nombre de processeurs, taille mémoire). Il est donc nécessaire de supporter ces trois algorithmes et, en fonction des paramètres caractérisant les opérandes et la configuration, choisir l'algorithme optimal. Ces algorithmes peuvent être implantés sur des microprocesseurs généraux, sans matériel spécial tels que (MENO81). Remarquons que l'algorithme décrit dans (MENO81) conduit aux mêmes performances que l'algorithme par produit, qui est plus simple.

2-2. Tri

La deuxième fonction des PJTs est le tri externe de grandes

relations, ainsi que le tri interne. Cette opération précède généralement la restitution des résultats à un usager. L'opération de tri est faite sur une relation en mémoire cache, et peut réaliser une projection sur certains attributs ayant servi au critère de tri. L'intérêt des tris pour les opérateurs relationnels, souligné dans (DEMO80) a été pris en compte au chapitre III, pour l'opérateur de jointure. Il y a été décrit notamment un algorithme multiprocesseur de tri-fusion d'ordre b, où b représente la capacité, à une page près, de la mémoire d'un processeur. Cet algorithme est simple et son développement ne nécessite pas de matériel spécial, ce qui est souvent le cas des algorithmes de tri multiprocesseurs (BANE79).

2-3. Semi-jointure

La troisième fonction des PJTs est la semi-jointure. Son intérêt, introduit dans (PALE72) et developpé dans (BERN79a) a inspiré notre étude d'algorithmes de semi-jointure au chapitre IV. La propriété de l'opérateur de semi-jointure est de restreindre les relations opérandes aux seuls tuples utiles à la réponse. De plus, c'est souvent un cas particulier de jointure, où les attributs résultats n'appartiennent qu'a une des deux relations. Il est montré au chapitre IV qu'il est souvent intéressant de combiner jointure et semi-jointures. Deux algorithmes multiprocesseurs de semi-jointure ont été proposés, l'un utilisant des tableaux de bits et l'autre des opérations de sélections réalisées par les filtres. L'algorithme par tableaux de bits sera implanté car il est plus général que l'algorithme par sélection et, dans le cas le plus défavorable, donne les mêmes performances. Un algorithme d'inequi-semi-jointure simple, utilisant les fonctions agrégats minimum et maximum, sera aussi implanté.

2-4. Elimination des doubles

Cette opération suit une opération de projection, telle que la traitent les filtres. La projection est exécutée à la vitesse du flot de données du disque par les filtres, qui génèrent ainsi des tuples en double. C'est une opération fréquente et importante qui peut être coûteuse, car elle nécessite de comparer chaque tuple de la relation avec chaque autre tuple. La méthode proposée est de précéder l'opération de dédoublage par une opération de tri, chaque tuple identique au précédent étant éliminé.

3. ALLOCATION DES PROCESSEURS DE JOINTURE ET TRI

L'allocation des PJTs et le contrôle de leur exécution parallèle est le rôle du noyau d'exécution répartie (NER). Plus généralement, le NER gère l'allocation et l'exécution répartie de tous les processeurs, et les coordonne. C'est un système opératoire, lui-même réparti, physiquement représenté par une couche de logiciel sur chacun des processeurs. Le NER est introduit au chapitre I en 4-4, et plus détaillé dans (BERN81). Nous résumons ici les concepts de NER afin de mettre en évidence les synchronisations des processeurs lors d'une exécution parallèle. Ceci est utile pour comprendre les spécifications des PJTs. Un plan d'exécution est composé d'une suite d'actions a exécuter chacune sur une classe de processeurs. Par exemple, l'action de sélection dans une relation permanente sera soumise aux filtres. Les actions sont déclenchées lorsque les conditions d'activation (ressouces disponibles, par exemple) ou de synchronisation (une opération terminée, par exemple) sont vérifiées. Une action est le déroulement d'un processus sur un processeur et peut engendrer des plans d'exécution composés aussi d'actions. Un plan d'exécution peut ne comporter qu'une seule action, qui aura alors une relative priorité par rapport aux autres. Ces actions isolées sont de type LIRE (une page), ECRIRE (une page) et doivent donc être exécutées très rapidement.

Au paragraphe 4-4 du chapitre I, nous avons considéré l'allocation de processeurs lors d'une requête demandant des opérations de sélection puis de jointure. Nous présentons maintenant plus en détail l'allocation des PJTs lors d'une opération de jointure. Premièrement, un plan d'exécution est soumis au NER par le processeur de manipulation de base, et précise une action de jointure et ses paramètres. Lorsque les variables de synchronisation, indiquant que les opérandes sont disponibles, sont vraies, le NER soumet l'action de jointure à un PJT, en ajoutant à la liste des paramètres le nombre disponible de PJTs à cet instant. Le processus ainsi activé par NER est un processus père. Comme plusieurs algorithmes de jointure sont possibles, le processus père en fonction de différents paramètres (nombre de PJTs, taille des relations à joindre, sélectivité des attributs) élit le meilleur algorithme avec le nombre de proceseurs optimal. Il construit alors un plan d'exécution, composé d'actions à effectuer en parallèle, par un ensemble de PJTs. Les actions génèreront autant de processus fils. Chaque processus fils reçoit un numéro logique consécutif. Lors de l'exécution parallèle d'un algorithme, une certaine synchronisation est nécéssaire entre les PJTs, c'est à dire entre processus père et fils. Le contrôle de l'exécution parallèle est le rôle du processus père et se fait à l'aide de variables externes gérées par le NER. Prenons l'exemple où l'algorithme par produit est choisi. Premièrement, une variable externe est initialisée à 0. Chaque processus fils activé demande des pages de la relation externe en soumettant des actions de lectures de pages au cache. Chaque processus ayant reçu une partie de la relation externe, ajoute 1 à la variable externe. Lorsque cette variable est égale au nombre de PJTs alloués à l'opération, le processus père est débloqué et demande la diffusion à tous les PJTs et lui-même d'une page de la relation interne. La variable externe est alors remise à 0. Lorsque la transmission est terminée, tous les processus (père et

fils) sont réactivés et réalisent la jointure interne des pages en mémoire. La variable externe est à nouveau mise a jour et le procédé se répète jusqu'à ce que toute la relation interne ait été lue. S'il reste des pages à traiter de la relation externe, le procédé est réinitialisé par le processus père. Lorsque toute la relation externe a ainsi été jointe à la relation interne, le processus père désactive tous les processus fils et récupère leur réponse qu'il communique au NER. Ainsi, pour une opération de jointure parallèle, le contrôle et l'allocation des processeurs est assuré par un quelconque des PJTs, qui exécute le processus père. Pour l'algorithme par produit, le rôle de la variable externe est de permettre la diffusion d'une même page à tous les PJTs en un seul transfert. Les services demandés par les PJTs lors d'une opération quelconque concernent la lecture de pages et l'écriture de pages en mémoire cache, au moyen d'actions adressées au NER.

Chaque processeur possède ainsi deux couches : les fonctions qu'il fournit, considérée comme application, et une partie NER. Cette partie NER est qualifiée de local par rapport au NER global, qui est implanté sur un ou plusieurs processeurs. Le NER global gère les files d'attente de plans d'exécution et communique avec les NER locaux. Lors des spécifications des PJTs, nous préciserons la partie fonction, étant entendu qu'une fonction est toujours appelée au travers de NER au moyen d'un plan d'exécution. La partie NER local est dupliquée sur chaque processeur. Pour plus de précisions, NER fait l'objet d'une étude approfondie dans (BERN81).

4. SPECIFICATIONS D'UN PROCESSEUR DE JOINTURE ET TRI

Les spécifications décrivent les interfaces externe et interne des PJTs au moyen du langage PASCAL. L'interface externe est l'ensemble des primitives appelées par le processeur de manipulation de base, ou en

d'autres termes, c'est le service offert par les PJTs. L'interface interne est l'ensemble des primitives exécutables en parallèle par les PJTs. Les primitives de l'interface interne sont appelées par une primitive de l'interface externe. Plus exactement, pour chaque primitive de l'interface externe correspondent plusieurs primitives de l'interface interne, supportant ainsi différents algorithmes pour réaliser la même opération. Les variables locales d'un PJT caractèrisent la mémoire locale du PJT et son numéro physique.

L'interface externe comprend les primitives suivantes :

- JOINDRE : jointure de deux relations,
- TRIER : tri d'une relation,
- DEDOUBLER : élimination des tuples dupliqués d'une relation,

L'interface interne comprend les primitives suivantes :

- JOINP : jointure par produit,
- JOINH : jointure par hachage,
- TRIE : tri,
- JOINT : jointure de deux relations triées,
- DDOUB : dédoublage d'une relation triée,
- SJOIN : semi-jointure par tableau de bits,
- ISJOIN : inequi-semi-jointure,

Ces interfaces avec le processeur de manipulation de base (PMB) et

les PJTs sont représentés figure 1.

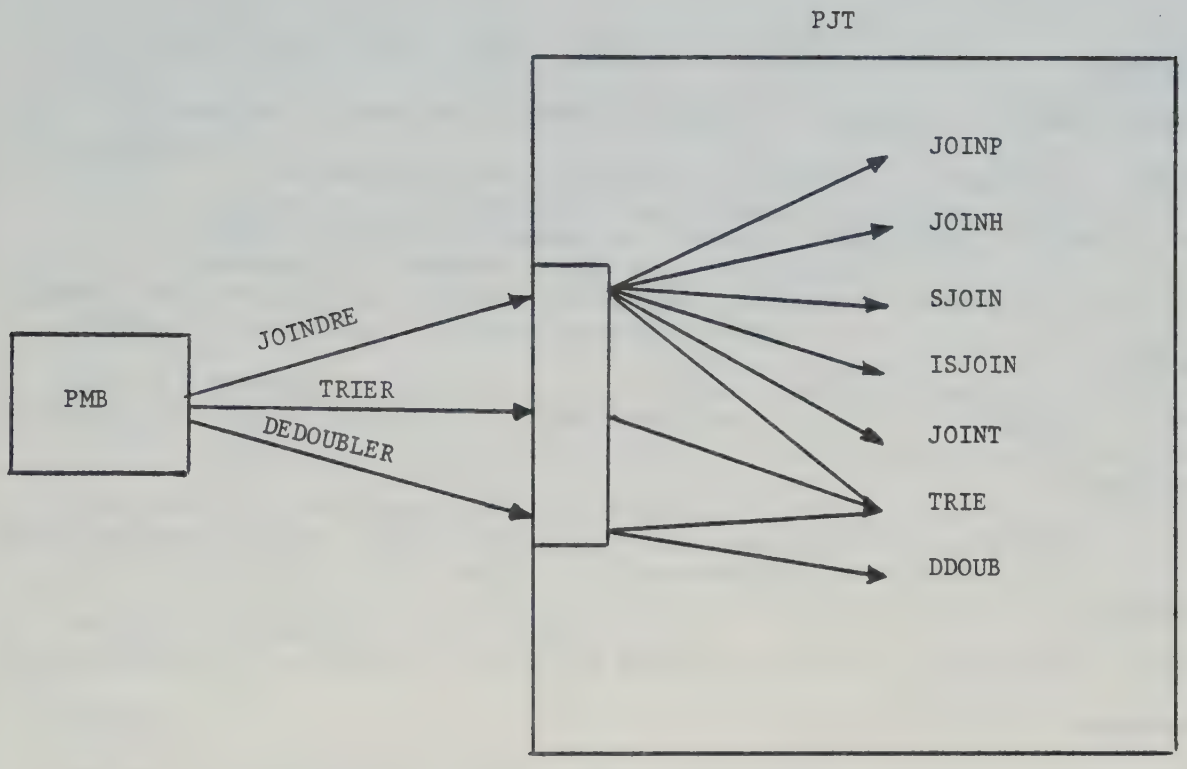


figure 1 : Vue générale des interfaces

4-1. Description de l'interface externe

Les variables locales d'un PJT sont :

```
var
    b : integer; (* parametre taille de mémoire propre *)
    C : integer; (* parametre taille de mémoire cache disponible
                    pour la jointure *)
    NP : integer ; (* no physique de processeur *)
```

Les primitives sont définies par des "function" PASCAL.

4-1-1. JOINDRE

function JOINDRE (IREL1, IREL2, IRELR, NBP1, NBP2, t1, t2, LAT, LCS, ATRI, CS1, CS2, NBPR)

paramètres :

IREL1, IREL2 : integer; (* numéros des relations a joindre *)

IRELR : integer; (* numéro de la relation résultat *)

NBP1, NBP2 : integer; (* nombre de pages de IREL1, IREL2 *)

t1, t2 : integer; (* nombre moyen de tuples par page de IREL1, IREL2 *)

LAT : LISTATT; (* liste des attributs à projeter dans la relation résultat *)

LCS : LISTCOND; (* prédicat de jointure sous forme conjonctive normale *)

ATRI : boolean; (* vrai si IRELR doit être triée sur l'attribut de jointure *)

CS1, CS2 : real; (* coefficients de sélectivité des semi-jointure *)

NBPR : integer; (* nombre de processeurs de jointure disponibles *)

JOINDRE : REPONSE; (* OK, NOK *)

fonction :

élire l'algorithme de jointure ou semi-jointure optimal en tenant compte des paramètres et activer les primitives de l'interface interne nécessaires afin de joindre les relations IREL1 et IREL2 dans la relation IRELR, en projetant éventuellement les tuples résultats sur LAT. Cette primitive détermine la séquence d'opérations optimale et le nombre de processeurs optimal par rapport à NBPR. Un plan d'exécution est alors constitué. S'il est décidé de précéder la jointure par des semi-jointures, le plan d'exécution indiquera une ou deux opérations de semi-jointures puis la jointure conditionnée par une variable de synchronisation (indiquant que les premiers résultats de semi-jointure

sont disponibles).

4-1-2. TRIER

function TRIER (IREL, NBP, IRELR, LAT, LCRI, NBPR)

paramètres :

IREL : integer; (* numéro de relation à trier *)

NBP : integer; (* nombre de pages de IREL *)

IRELR : integer; (* relation résultat *)

LAT : LISTATT; (* liste des attributs pour projection *)

LCRI : CRITRI; (* critère de tri = liste d'attributs avec le sens (ascendant, descendant) par attribut *)

NBPR : integer; (* nombre de processeurs disponibles *)

TRIER : REPONSE; (* OK, NOK *)

fonction :

trier la relation IREL sur le critère LCRI en projetant éventuellement sur LAT lors de la dernière passe de tri. Le nombre optimal de processeurs, en fonction de NBPR, est alloué à l'opération de tri. Si NBP$\leq$b, la relation peut être contenue en mémoire propre et le processeur exécutant la primitive TRIER réalise lui-meme le tri interne de la relation. Si NBP$>$b, un plan d'exécution est composé avec un nombre optimal d'actions parallèles TRIE. A chaque étape de tri, une relation de travail est créée pour écrire la relation partiellement triée. Ainsi, p processus exécutant les opérations TRIE réalisent des fusions d'ordre b jusqu'a la phase optimale, qui génère p monotonies. A la phase optimale, p/b processus sont désactivés à chaque nouvelle passe de la phase post-optimale, de sorte que la dernière fusion soit exécutée par le processus père.

4-1-3. DEDOUBLER

function DEDOUBLER (IREL, NBP, IRELR, LTYPA, NBPR)

paramètres :

IREL : integer; (* numéro de relation à dédoubler *)

NBP : integer; (* nombre de pages de IREL *)

IRELR : integer; (* relation résultat *)

LTYPA : LISTTYPATT; (* liste des types d'attributs d'un tuple, ce qui permet de trier sur chacun des attributs par ordre croissant *)

NBPR : integer; (* nombre de processeurs disponibles *)

DEDOUBLER : REPONSE; (* OK, NOK *)

fonction :

éliminer les tuples dupliqués de la relation IREL et écrire le résultat du dédoublage dans la relation IRELR. Un critère de tri est constitué à partir de LTYPA et la primitive TRIER est appelée. Puis, un plan d'exécution composé de plusieurs appels à DDOUB est construit.

4-2. Interface interne

L'interface interne décrit l'ensemble des primitives exécutées en parllele Ces primitives sont appelées par des primitives de l'interface externe.

(1) Function JOINP (IREL1, IREL2, NBP1, NBP2, IRELR, LAT, LCS, K, P);

parametres :

IREL1, IREL2 : integer; (* relations à joindre *)

NBP1, NBP2 : integer; (* nombre de pages de IREL1, IREL2 *)

IRELR : integer; (* relation résultat *)

LAT : LISTATT; (* liste des attributs résultats à projeter *)

LCS : LISTCOND; (* prédicat de jointure *)

K : integer; (* numero relatif de processeur pour cette opération *)

P : integer; (* nombre de processeurs alloués à cette opération *)

JOINP : REPONSE; (* OK, NOK *)

fonction :

exécuter en parallele avec p-1 autres processeurs l'algorithme de jointure par produit des relations IREL1 et IREL2 dans la relation résultat IRELR. Cette primitives demande des lectures et ecritures de pages en mémoire cache. Le processeur K lit b-1 pages de IREL1 en mémoire locale, et, pour chaque page de IREL2 lue, compare tout tuple de IREL1 à tout tuple de IREL2 suivant le prédicat LCS. Lorsque le prédicat est vérifié, la concaténation des deux tuples et leur projection suivant LAT est écrite dans le tampon résultat. Ce tampon est écrit dans la relation IRELR lorsqu'il est plein.

(2) Function TRIE (IREL, IRELR, LAT, LCRI, NO1P, TAILM, K, P);

parametres :

IREL : integer; (* relation à trier *)

IRELR : integer; (* relation résultat *)

NO1P : integer; (* numéro de la 1ère page a lire *)

TAILM : integer; (* taille des monotonies a fusionner en nombre de pages *)

LAT : LISTATT; (* liste d'attributs a projeter *)

LCRI : CRITRI; (* critère de tri *)

K : integer; (* numero relatif de processeur pour cette opération *)

P : integer; (* nombre de processeurs alloués à l'opération *)

TRIE : REPONSE; (* OK, NOK *)

fonction :

exécuter en parallèle avec P-1 processeurs une étape de tri fusion d'ordre b. b monotonies sont lues, une page à la fois, en mémoire par des appels de lectures à partir de la page NO1P et fusionnées en utilisant une page de mémoire. Le processus père exécutant TRIER fait $\log_b$ NBP appels à TRIE (par plusieurs processeurs). A chaque nouvelle étape, une relation de travail est créée en mémoire cache pour contenir le résultat de l'étape de tri.

(3) Function JOINT (IREL1, IREL2, IRELR, LAT, LCS);

paramètres :

IREL1, IREL2 : integer: (* relations triées a joindre *)

IRELR : integer; (* relation résultat *)

LAT : LISTATT; (* liste d'attributs a projeter *)

LCS : LISTCOND; (* prédicat de jointure *)

JOINT : REPONSE; (* OK, NOK *)

fonction :

exécuter la jointure par fusion de deux relations triées IREL1 et IRELR sur l'attribut de jointure dans la relation résultat IRELR. Un seul processeur est nécéssaire pour cette opération, qui demande des lectures et écritures séquentielles de pages.

(4) Function JOINH (IREL, IRELH, NBPA, IRELR, LAT, LCS, TBOOL);

parametres :

IREL : integer; (* relation à répartir *)

IRELH : integer; (* fichier haché *)

NBPA : integer; (* nombre de paquets du fichier haché *)

IRELR : integer; (* relation résultat *)

LAT : LISTATT; (* liste d'attributs pour la projection *)

LCS : LISTCOND; (* prédicat de jointure *)

TBOOL : TABBOOL; (* tableau de bits associé à l'opération *)

JOINH : REPONSE; (* OK, NOK *)

fonction :

appliquer l'algorithme par hachage des relations IREL et IRELH. Le tableau TBOOL a été mis a jour en mémoire cache. La relation IREL est lue séquentiellement et les blocs de la relation IRELH sont accédés lorsque tout bit correspondant à une valeur d'attribut de jointure est marqué dans TBOOL.

(5) Function DDOUB (IREL, IRELR, LAT, K, P);

parametres :

IREL : integer; (* relation a dédoubler *)

IRELR : integer; (* relation résultat *)

LAT : LISTATT; (* liste d'attributs d'un tuple *)

K : integer; (* numéro de processeur logique *)

P : integer; (* nombre de processeurs alloués a l'opération *)

fonction :

éliminer les tuples en double de la relation IREL triée. L'opération est réalisée en parallèle avec p-1 autres processeurs.

(8) Function SJOIN (IREL1, IREL2, IRELR, TBOOL, LAT, LCS);

parametres :

IREL1 : integer; (* relation source (à répartir *)

IREL2 : integer; (* relation contenant la liste triée des différentes valeurs d'attributs de jointure *)

IRELR : integer; (* relation résultat *)

TBOOL : integer; (* numéro de tableau de bits mis à jour *)

LAT : LISTATT; (* liste d'attributs pour projection *)

LCS : LISTCOND; (* prédicat d'equi-semi-jointure *)

SJOIN : REPONSE; (* OK, NOK *)

fonction :

réaliser la semi-jointure par tableaux de bits de la relation IREL1 par la relation IREL2 suivant le prédicat LCS. La relation IREL2 contient la liste triée sans doubles des valeurs d'attributs de jointure (réalisée par DEDOUBLER). Le tableau de bits TBOOL a été mis à jour au cours d'une opération demandée au gérant du cache. La relation IREL2 et le tableau TBOOL sont lus en mémoire locale, et y résident pendant toute l'exécution de la function SJOIN.

(7) Function ISJOIN (IREL1, IREL2, IRELR, LAT, LCS)

parametres :

```
IREL1 : integer; (* relation source *)
IREL2 : integer; (* relation cible *)
IRELR : integer;  (* relation résultat *)
LAT : LISTATT;   (* liste d'attributs de la relation IREL1 à
projeter *)
LCS : LISTCOND;  (* prédicat de jointure *)
ISJOIN : REPONSE; (* OK, NOK *)
```

fonction :

réaliser l'inequi-semi-jointure de la relation IREL1 par la relation IREL2 sur le prédicat LCS. Cette primitive appelle les fonctions Minimum et Maximum, et la fonction SELECTER offertes par les filtres.

5. SERVICES DEMANDES

Les services sont demandés par les PJTs uniquement au gérant de la mémoire cache au moyen de plans d'exécution composés généralement d'actions isolées prioritaires. Ces services concernent essentiellement l'acquisition des pages opérandes, la restitution de pages résultats et la gestion de tableaux de bits et relations temporaires. Vu par les

PJTs, le gérant du cache rend un service de système de gestion de fichiers, où le granule d'information échangé est la page. Ainsi, l'adressage par les PJTs consiste à nommer la relation référencée et le numéro de page. La conversion en adresse absolue est le rôle du gérant du cache. De plus, il gère des pointeurs courants par relation. Il peut créer et détruire à la demande des relations temporaires, utiles , par exemple, pour fournir des relations de travail lors d'un tri. D'autre part, un filtre peut être associé à la mémoire cache pour assurer la fonction de sélection. Ce service de sélection peut donc être demandé au gérant du cache par les PJTs. Nous décrivons maintenant les primitives d'appel du gérant de la mémoire cache, qui rendent le service demandé par les PJTs.

5-1. Création et destruction de relations temporaires

La création de relation temporaire est demandée avant l'exécution de tout algorithme nécessitant une relation de travail, généralement pour écrire un résultat intermédiaire.

function CRETRA (var IREL:integer) : REPONSE;

crée une relation de travail en mémoire cache, c'est à dire une entrée correspondante dans la table des relations référencées. IREL est le numéro de la relation créée attribué par le gérant du cache. Il permettra au PJT de nommer la relation.

function DETRA (IREL:integer) : REPONSE;

détruit la relation IREL gérée en mémoire cache et précédemment créée par CRETRA.

5-2. Lecture de relation

Ce service est demandé lors de toute opération afin de charger la mémoire locale des PJTs. Deux méthodes d'accès aux pages sont nécéssaires ; relative et séquentielle. Une relation de N pages est vue par les PJTs comme un ensemble de pages numérotées et chainées de 0 à N-1. L'adressage relatif d'une page consiste à spécifier son numéro. Ainsi, plusieurs processeurs peuvent demander la même page, ce qui est le cas pour la diffusion de la relation interne dans l'algorithme de jointure par produit. La méthode d'accès séquentiel permet de répartir une relation sur plusieurs processeurs tout en prévenant des mêmes lectures de pages par les différents processeurs, par exemple, lors de la répartition de la relation externe dans l'algorithme de jointure par produit. Ceci est réalisé par le gérant du cache en maintenant un pointeur courant par relation, modifié de manière indivisible. Deux fonctions sont fournies : LIRED et LIRES. Le code REPONSE indique OK, NOK ou FIN RELATION.

function LIRED (IREL, NOPA: integer; var ADP :PAGE) : REPONSE;

lire directement la page NOPA de la relation IREL, et obtenir le résultat à ADP, adresse en mémoire locale.

function LIRES (IREL:integer; var ADP: PAGE) : REPONSE;

lire la page de la relation IREL pointée par le pointeur courant et obtenir le résultat a ADP, adresse en mémoire locale.

5-3. Ecriture de relation

Comme pour la lecture deux méthodes d'accès sont encore nécéssaires : séquentielle et relative. L'accès sequentiel est utilisé lorsque l'ordre des pages produites par les PJTs n'importe pas. L'accès relatif est utilisé lors des opérations de type tri pour garantir l'ordre des pages dans les monotonies. Les fonctions fournies sont ECRIRES et

ECRIRED. Le code REPONSE est OK ou NOK.

function ECRIRES (IREL:integer; ADP: PAGE) : REPONSE;

écrire la page qui se trouve à ADP en mémoire locale dans la relation IREL à la page suivant le pointeur courant.

function ECRIRED (IREL, NOPA:integer; ADP:PAGE) : REPONSE;

écrire la page à ADP en mémoire locale dans la relation IREL à la page NOPA.

5-4. Manipulation de tableaux de bits

Les tableaux de bits permettent d'éviter des accès inutiles à un ensemble de données, lors de la deuxième phase de l'algorithme de jointure par hachage ou de l'algorithme de semi-jointure. Il faut donc pouvoir créer, détruire, mettre a jour ou lire un tableau de bits, géré en mémoire cache. Le code REPONSE est OK ou NOK.

function CREEB (N: integer; var B: integer) : REPONSE;

initialiser un tableau de N bits , dont le numéro sera rendu dans B.

function DETRUIRB (B: integer) : REPONSE;

détruire le tableau B en mémoire cache.

function MAJB (IREL, B : integer; LAT: LISTATT) : REPONSE;

mettre à jour le tableau de bits B à partir des attributs indiqués dans LAT dont les valeurs sont contenues dans la relation IREL.

function LIREB (B: integer; var ADP:PAGE) : REPONSE;

lire le tableau de bits B dans la page ADP de mémoire locale.

5-5. Hachage d'une relation en mémoire cache

Ce service est nécessaire pa l'algorithme de jointure utilisant des

tableaux de bits. Il consiste à hacher une relation sur les attributs de jointure et, simultanément, mettre à jour un tableau de bits.

function HACHER (IREL, IRELH, N , B : integer; CLEH : LISTATT) : REPONSE;

hacher la relation IREL dans la relation IRELH en mémoire cache sur la clé de hachage CLEH. La relation sera hachée dans N blocs, et simultanément, le tableau B sera mis à jour.

5-6. Services demandés aux filtres

Ces services sont demandés par l'intermédiaire du gérant du cache et concernent les opérations de projection, et les fonctions aggrégats Minimum et Maximum. Ces services sont nécessaires lors des opérations d'equi et inequi-semi-jointures.

function SELECTER (IREL, IRELR: integer; LAT: LISTATT) :REPONSE;

projeter la relation IREL dans la relation IRELR sur les attributs indiqués dans LAT.

function MIN (IREL: integer; LAT: LISTATT; var LATR: LISTATTR) : REPONSE;

calcul de la valeur minimum dans la relation IREL des attributs indiqués dans LAT, le résultat étant écrit dans LATR.

function MAX (IREL: integer; LAT: LISTATT; var LATR: LISTATTR): REPONSE;

calcul de la valeur maximum dans la relation IREL des attributs indiqués dans LAT, le résultat étant écrit dans LATR.

6. CONCLUSION

Dans ce chapitre, les spécifications d'une classe de processeurs spécialisés , les processeurs de jointure et tri, pour la machine bases de données SABRE ont été décrites à l'aide du langage PASCAL. Chaque PJT supporte des algorithmes parallèles de jointure, tri, semi-jointure et d'élimination des doubles. L'interface externe des PJTs décrit les services rendus au processeur de manipulation de base, qui opère la décomposition et optimisation de question. L'interface interne des PJTs décrit les primitives nécéssaires à l'exécution parallèle d'une primitive de l'interface externe. La stratégie d'allocation des processeurs est réalisée par un noyau d'exécution réparti (NER), qui est implanté localement sur chaque processeur. Le NER offre les outils nécéssaires au controle et à la synchronisation de processus s'exécutant en parallèle sur différents processeurs. Ceci a permis de décrire les interfaces en PASCAL, indépendamment des synchronisations et du parallélisme, qui ne peuvent être simulés par ce langage. Enfin, les services demandés par les PJTs au gérant de la mémoire cache ont été présentés par la liste de leurs primitives. Ces services concernent l'acquisition et la restitution de pages en mémoire cache ainsi que la gestion de tableaux de bits.

Ces interfaces, clairement définis, permettent la réalisation d'un PJT en PASCAL qui s'intègre dans une première version monoprocesseur de SABRE. Cette version a pour objectif de valider les interfaces et les algorithmes monoprocesseurs. Les jeux de tests du programme PASCAL sont décrits en annexe 2. Actuellement, les trois principales fonctions du PJT sont réalisées : jointure, tri et élimination des doubles. L'algorithme de jointure implanté est l'algorithme par produit.

CONCLUSION

Dans cette étude, nous avons présentés et analysés des algorithmes multiprocesseurs pour réaliser des opérations sur des bases de données relationnelles. L'intérêt a porté sur les opérations qui nécessitent un traitement répétitif sur des relations. Ces opérations sont jointure, tri, semi-jointure et élimination des doubles. Le contexte d'étude de ces algorithmes a été le projet de machine bases de données SABRE. Un modèle d'architecture multiprocesseur typique des machines bases de données en cours de développement a permis d'isoler une méthode d'analyse des algorithmes. Les critères d'analyse retenus ont été le coût d'entrée-sortie, le coût de traitement CPU, et le coût de transmission de messages. L'hypothèse majeure a été que les relations sont trop grandes pour être entièrement contenues en mémoire. Trois algorithmes multiprocesseurs de jointure ont été proposé. L'analyse de ces algorithmes a conduit aux constatations suivantes. L'algorithme par produit, le plus simple, se comporte le mieux avec un grand nombre de processeurs et réalise aisément l'inequi-jointure. L'algorithme de jointure utilisant un tri ascendant multiprocesseur est efficace pour traiter de grandes relations opérandes. De plus, la relation résultat est triée sur l'attribut de jointure. L'algorithme par hachage est performant lorsque la proportion de tuples participant à la jointure est faible. Finalement, il a été montré que chacun des algorithmes possède un créneau d'application défini par la configuration et les caractéristiques des relations à joindre. En conséquence, il est souhaitable d'implanter ces trois algorithmes, avec la possibilité d'en élire un en fonction de la valeur des paramètres de la requête demandée.

L'intérêt de l'opérateur de semi-jointure pour réduire le coût de communication lors des opérations binaires dans un système multiprocesseur a été souligné. Dans un tel contexte, deux algorithmes d'equi-semi-jointure et un algorithme d'inequi-semi-jointure ont été présentés et analysés. Ils s'exécutent en des temps linéairement

proportionnels au volume des relations opérandes et résultats, et inversement proportionnels au nombre de processeurs. Il a été montré que précéder l'opération de jointure par des semi-jointures était plus efficace que la jointure des relations. Il est donc important ,pour une machine bases de données bien spécifiée, de supporter cet opérateur.

Finalement, les spécifications d'un processeur spécialisé assigné aux fonctions de jointure, tri, semi-jointure et élimination des doubles ont été décrites par un ensemble de primitives exprimées en langage PASCAL. La méthode proposée pour éliminer les tuples en double d'une relation est basée sur l'opérateur de tri. Ces spécifications permettent l'intégration d'un processeur de jointure et tri dans une version SABRE monoprocesseur, dont le but est la validation des interfaces et des algorithmes. Une première version d'un processeur de jointure et tri est disponible à l'INRIA, sous le système MULTICS, programmée en langage PASCAL. Ce processeur (fonctionnel) remplit les fonctions de jointure, tri et élimination de doubles. Actuellement, l'algorithme de jointure par produit est implanté. Cette version sera complétée par deux autres algorithmes de jointure. Une prochaine version devrait permettre la validation des algorithmes parallèles proposés sur un système multi-microprocesseurs.

Cette étude peut être étendue dans plusieurs directions. Premièrement, une plus exacte détermination des coûts de ces algorithmes pourrait être obtenue par modélisation et mesures de performances. Il est probable que le paramètre taux d'efficacité de la mémoire cache, supposé identique pour les algorithmes, dépende des algorithmes développés. D'autre part, il serait intéressant de prendre en compte le coût de contrôle des processeurs lors d'un algorithme complexe, tel que le tri. Cependant, le coût dominant resterait le temps d'entrée-sortie.

Deuxièmement, d'autres méthodes pourraient être étudiées. Des méthodes connues consistent à utiliser des index non ordonnés, triés ou hachés. Ces méthodes pourraient être basées sur les mêmes principes que l'algorithme par produit, tri et hachage, excepté qu'elles travailleraient sur des clés. Les performances de ces méthodes dépendraient essentiellement du placement des données dans l'espace disque et de leur bonne adaptation à l'exécution parallele.

Jusqu'à présent, nous avons évoqué la réalisation des processeurs de jointure et tri avec des microprocesseurs généraux. Une autre possibilité intéressante serait l'implantation des opérations parallèles de jointure et tri en utilisant les technologies LSI ou VLSI. Pour cela, il faudrait définir une organisation spéciale de processeurs et mémoires, et spécifier la combinaison optimale de processeurs et mémoires à placer sur une carte. Les algorithmes proposés dans notre étude pourraient aisément y être implantés.

Finalement, les performances de chacun des algorithmes devraient être analysée dans le contexte d'une question complete. Le choix d'un algorithme pour une opération serait conditionné par les autres opérations de la question. L'analyse pourrait être aussi étendue au cas où plusieurs requêtes sont exécutées simultanément, afin de considérer et mesurer le débit global du système.

L'application des solutions proposées dans notre étude se limite aux bases de données alphanumériques structurées. Les récents progres technologiques pour traiter les images et les textes vont conduire au développement de systèmes de gestion de bases d'informations (SGBI), intégrant à la fois des données alphanumériques, textes et images (TANG81). Une donnée image représente une donnée qui peut être affichée à l'aide d'un terminal spécifique. Des exemples de données images sont une signature, une photographie, une carte géographique etc. Les

données textes et images, par leurs structures floues, sont plus difficiles a maitriser que les données alphanumériques. L'extension du modèle relationnel devrait permettre d'accepter ces types de données. Il serait alors utile de définir un opérateur similaire à la jointure, capable de comparer et d'associer des données de même type, ayant des propriétés communes. La jointure de deux textes permettrait d'associer des sous-ensembles de textes (paragraphes) avant un ensemble de mots en commun. La jointure de données images serait aussi une opération puissante. Elle permettrait, par exemple, d'obtenir des images entières à partir de différentes parties d'images. L'opération pourrait être étendue à l'intersection de deux objets (par exemple, deux ensembles de courbes) avec éventuellement une condition sur la forme de l'intersection. Il est évident que cette opération serait très couteuse en temps d'exécution.En supposant l'extension des machines bases de données à la gestion de bases d'informations, une voie prometteuse de future recherche serait la conception d'algorithmes parallèles de jointure de données alphanumériques, textes et images.

REFERENCES

ADIB78 M. ADIBA, C. CHUPIN, R. DEMOLOMBE, G. GARDARIN, J. LEBIHAN "Issues in Distributed Database Management Systems : a Technical Overview" Proc. of the 4th International Conference on Very Large Data Bases, Berlin, 1978, SIRIUS Report.

AHO 79 A.V. AHO, G. BEERI, J.D. ULLMAN "The Theory of Joins in Relational Databases" ACM Transactions on Data Bases Systems, vol.4, no3, september 1979.

ASTR76 M. ASTRAHAN et al. "System-R : a Relational Approach to Database Management" ACM Transactions on Database Systems, vol.1, no2, june 1976, pp 97-137.

AUER80 H. AUER et al. "R.D.B.M : a Relational Data Base Machine" Internal Report, Technisch Universitat Braunchweig, nr8005, june 1980.

BABB79 E. BABB "Implementing a Relational Data Base by Means of Specialized Hardware" ACM Transactions on Database Systems, vol.4, no1, march 1979, pp 1-29.

BAER80 J.L. BAER, G. GARDARIN, C. GIRAULT, G. ROUCAIROL "The Two-Step Commitment Protocol : Modeling, Specification and Proof Methodology" 5th International Conference on Software Engineering, San Diego, 1981.

BANC80 F. BANCILHON, M. SCHOLL "Design of a Back-end Processor for a Database Machine" Proceedings of the ACM-SIGMOD, Santa Monica, California, may 1980.

BANE78 J. BANERJEE, D.K. HSIAO, R.I. BAUM "Concepts and Capabilities of a Database Computer" ACM Transactions on Database Systems, vol.3, no.4, december 1978, pp 347-384.

BANE79 J. BANERJEE, D.K. HSIAO "Parallel Bitonic Record Sort : an Effective Algorithm for the Realization of a Post Processor" Technical Report, OSU-CISRC-TR-79-1, Computer and Information Science Research Center, Columbus, Ohio, 1979.

BELL80 Z. BELLAHSENE "Optimisation du Traitement des Requetes dans une Machine Bases de Données Relationnelle" Rapport de DEA, Institut de Programmation, PARIS VI, septembre 1980.

BERN79a P.A. BERNSTEIN, D.M. CHIU "Using Semi-Joins to Solve Relational Queries" Technical Report, CCA-01-79, january, 1979.

BERN79b P.A. BERNSTEIN, N. GOODMAN "Full Reducers for Relational Queries Using Multi-Attribute Semi-Join" Proc. 1979 NBS Symposium on Computer Network, december 1979.

BERN79c P.A. BERNSTEIN, N. GOODMAN
"Inequality Semi-Joins" Technical Report, CCA-79-28, december 1979.

BERN80 P.A. BERNSTEIN, N. GOODMAN
"Timestamp-Based Algorithm for Concurrency Control in Distributed Database Systems" 6th International Conference on Very Large Data Bases, Montreal, 1980, Proc. IEEE, 285-300.

BERN81 P. BERNADAT
"La Machine Bases de Données SABRE et son Novau d'Exécution Répartie (NER)" Rapport de DEA, Institut de Programmation, PARIS VI, septembre 1981.

BLAS77 M.W. BLASGEN, K.P. ESWARAN
"Storage and Access in Relational Databases" IBM System Journal, vol.16, no.4, 1977, pp 363-378.

BORA80 H. BORAL, D.J. DEWITT, D. FRIEDLAND, W.K. WILKINSON
"Parallel Algorithms for the Execution of Relational Operations" Technical Report 402, Computer Science Department, University of Wisconsin, Madison, janvuary 1980.

BORA81 H. BORAL, D.J. DEWITT, W.K. WILKINSON
"Performances Evaluation of Associative Disk Designs" 6th Workshop on Architecture for Non Numeric Processing, Hyeres, France, june 1981.

BRAT80 K. BRATBERGSENGEN, R. LARSEN, O. RISNES, T. AANDALEN
"A Neighbor Connected Processor Network for Performing Relational Algebra Operations" Department of Computer Science, University of Trondheim (Norway), 1980.

CHAM91 D.D. CHAMBERLIN, A.M. GILBERT, R.A. YOST
"A History of System-R and SQL/ Data System" 7 th Int. Conf. on Veru Large Data Bases, september 1981, Cannes, France.

CHIU80 D.M. CHIU, Y.C. HO
"A Methodology for Interpreting Tree Queries into Optimal Semi-Join Expressions" ACM Transactions on Database Systems, 18, december 1980, pp 169-178.

CHU 80 G. GARDARIN, W. CHU
"A Distributed Control Algorithm for Reliability and Consistency Updating Replicated Databases" IEEE Transactions on Computers, Vol. C.29/N12, December 1980.

CODD70 E.F. CODD
"A Relational Model of Data for Large Shared Data Banks" CACM 13, june 1970, pp 377-387.

DATE75 C.J. DATE
"An Introduction to Database Systems" Addison-Weslay, may 1975.

DEMI78 R.A. DEMILLO, D.P. DOBKIN, A.K. JONES, R.J. LIPTON
"Foundation of Secure Computation" Accademic Press 1978.

DEMO80 R. DEMOLOMBE
"Intérêt des Processeurs de Tri dans les SGBD Relationnels" Journées Machines Bases de Données, Sophia-Antipolis, France, septembre 1980.

DEWI79 D.J. DEWITT
"Query Execution in Direct" Proceedings of the ACM-SIGMOD 1979, International Conference of Management of Data, may 1979,pp 13-22.

DEWI80 H. BORAL, D.J. DEWITT
"Design Considerations for Data-Flow Database Machines" Proceedings of the ACM-SIGMOD, Conference of Management of Data, Los Angeles, 1980, pp 94-104.

DEWI81 D.J. DEWITT, P.B. HAWTHORN
"A Performance Evaluation of Database Machine Architectures" 7th Int. Conf. on Very Large Data Bases, september 1981, Cannes, France.

ESWA76 K.P. ESWARAN, J.N. GRAY, R.A. LORIE, I.L. TRAIGER
"The Notions of Consistency and Predicate Locks in a Database System" Comm. ACM, vol. 19/N11, november 1976, pp624-633.

FAUD80 P. FAUDEMAY
"Sur une Nouvelle Classe de Filtres Multi-Expressions" Journees Machines Bases de Donnees, Sophia Antipolis, septembre 1980, INRIA.

GARD77 G. GARDARIN, P. LEBEUX
"Scheduling Algorithms for Avoiding Inconsistency in Large Databases" 3rd International Conference on Very Large Data Bases, Tokyo, 1977, Proc. IEEE, pp 501-506.

GARD79a G. GARDARIN
"A Unified Architecture for Data and Message Management" National Computer Conference, New-York, 1979, AFIPS vol. 48, pp 681-688.

GARD79b G. GARDARIN, M. MELKANOFF
"Proving Consistency of Database Transactions" 5th International Conference on Very Large Data Bases, Rio, 1979, Proc. IEEE, pp 291-298.

GARD80a G. GARDARIN et al.
"Objectifs, Principes et Architecture d'une Machine Bases de Donnees Réparties" Journées Machines Bases de Données, Sophia-Antipolis, France, 1980.

GARD80b G. GARDARIN, J. BEGUE
"Un Protocole de Manipulation de Données Relationnel" Congres AFCET 1980, Nancy, France.

GARD80c G. GARDARIN
"Integrity, Consistency, Concurrency, Reliability in Distributed Database Management Systems" Distributed Data Bases, North-Holland 1980, pp 335-351.

GARD81 G. GARDARIN
"An Introduction to SABRE : a Multi-Microprocessor Database Machine" 6th Workshop on Computer Architecture for Non Numeric Processing, Hyeres, France, june 1981.

GOTL75 L.R. GOTLIEB
"Computing Joins of Relations" Proceedings of the ACM-SIGMOD, Conference of Management of Data, San Jose, California, mav 1975, pp 55-63.

GRIF76 P.P. GRIFFITHS, B.W. WADE
"An Authorization Mechanism for a Relational Database System" ACM Transactions on Database Svstems, vol.1/N3, pp 242-255, september 1976.

ISO 79 ISO/TC97/SC16 N227
"Reference Model of Open System Interconnection" June 1979.

INTE79 INTEL
"Les Concepts Systemes d'Intel : le Multibus et ses Signaux" E.A.I.263, A. Sabatier, Intel-France, fevrier 1979.

KARL80 K. KARLSSON
"Le Modele d'Acces de la Machine Bases de Données SABRE" Journées Machines Bases de Données, INRIA, Sophia-Antipolis, France, septembre 1980.

KARL81 K. KARLSSON
"Reduced Cover-Trees and their Application in the SABRE Access Path Model" 7th Int. Conf. on Very Large Data Bases, september 1981, Cannes, France.

KNUT73 D.E. KNUTH
"The Art of Computer Programming" Vol.3, Sorting and searching, Reading, MA : Addison-Wesley, 1973.

KUBR81 G. KUBRYCK
"Analyse des Systemes de Mémoires a Bulles Magnétiques dans le Contexte SABRE" Rapport de DEA, Institut de Programmation, PARIS VI, septembre 1981.

LEBI80 J. LEBIHAN et al.
"SIRIUS : a French Nationwide Project on Distributed Data Bases" 6th International Conference on Very Large Data Bases, Montreal, 1980, Proc. IEEE.

LIN76 C.S. LIN, D.C.P. SMITH, J.M. SMITH
"The Design of Rotating Associative Memory for Relational Database Applications" ACM Transactions on Database Systems, march 1976, pp 53-65.

LIPO75 G.J. LIPOVSKI, S.Y.W. SU
"CASSM : a Cellular System for Very Large Data Bases" 1st International Conference on Very Large Data Bases, september 1975, pp 456-472.

MENO81 J.M. MENON, D.K. HSIAO
"Design and Analysis of Relational Join Operations of a Database Computer (DBC)" 7th Int. Conf. on Very Large Data Bases, september 1981, Cannes, France.

OFLA80 K. OFLAZER, E.A. OZKARAHAN
"RAP.3 : a Multi-Microprocessor Cell Architecture for the RAP Database Machine" Proc. of the International Workshop on High-Level Language Computer Architecture, Fort Lauderdale, Florida, may 1980.

OZKA75 E.A. OZKARAHAN, S.A. SCHULSTER, K.C. SMITH
"RAP : Associative Processor for Data Base Management" Proc. AFIPS 1975 NCC, vol.44, AFIPS Press, Montvale, N.J.,pp 379-388.

OZKA77 E.A. OZKARAHAN, S.A. SCHULSTER, K.C. SEVCIK
"Performance Evaluation of a Relational Associative Processor" ACM Transactions on Database Systems, june 1977, pp 175-196.

ROTH80 J.B. ROTHNIE et al.
"SDD1 : a System for Distributed Databases" ACM Transactions on Database Systems, march 1980, pp 1-17.

PALE72 F.P. PALERMO
"A Data Search Problem" Proc. of the 4th International Symposium on Computer and Information Science, Miami beach, Florida, december 1972.

PECH75 R.M. PECHERER
"Efficient Evaluation of Expressions in a Relational Algebra" Proc. of the ACM Pacific 75 Regional Conference, April 1975, New-York, pp 44-49.

PREP78 F.P. PREPARATA
"New Parallel-Sorting Schemes" IEEE Transactions on Computers, vol.C-27, july 1978.

SELI79 P.G. SELINGER, M.M. ASTRAHAN, D.D. CHAMBERLIN, R.A. LORIE, T.G. PRICE
"Access Path Selection in a Relational Database Management System" ACM-SIGMOD 1979, Boston, 1979, pp 23-34.

SMIT75 J.M. SMITH, P.E. CHANG
"Optimizing the Performance of a Relational Algebra Database Interface" CACM, vol.18, no10, october 1975, pp 133-155.

STON76 M. STONEBRAKER, E. WONG, P. KREPS
"The Design and Implementation of INGRES" ACM Transactions on data base systems, vol.1, no.3, september 1976.

STON79 M. STONEBRAKER
"Muffin : a Distributed Data Base Machine" First International Conference on Distributed Computing Systems, 1979, pp 459-469.

TANG81 G.Y. TANG
"A Management System for an Integrated Database of Pictures and Alphanumerical Data" Computer Graphics and Image Processing, 16, 1981, pp 270-286.

TEMM81 N. TEMMERMAN
"Un Protocole de Manipulation de Données pour la Machine Bases de Données SABRE" Rapport de DEA, Institut de Programmation, PARIS VI, septembre 1981.

VALD80 G. GARDARIN, P. VALDURIEZ
"Algorithmes Multiprocesseurs de Jointure de Relations" Journées Machines Bases de Données, INRIA, Sophia-Antipolis, France, septembre 1980.

VALD81a P. VALDURIEZ
"Algorithmes de Jointures de Relations" Colloque "Les Bases de Donnees", AFCET, Tunis, avril 1981.

VALD81b P. VALDURIEZ, G. GARDARIN
"Multiprocessor Join Algorithms of Relations" Rapport Interne SABRE, INRIA, aout 1981.

VALD81c P. VALDURIEZ
"Semi-Join Algorithms for Multiprocessor Systems" Rapport Interne SABRE, INRIA, septembre 1981.

VIEM81 Y. VIEMONT
"Concurrence et Fiabilité dans la Machine Bases de Données SABRE" Rapport de DEA, Institut de Programmation, PARIS VI, septembre 1981.

YAO 79 S.B. YAO
"Optimization of Query Evaluation Algorithms" ACM Transactions on Database Systems, vol.4, no2, june 1979, pp 133-155.

WAH 80 B.W. WAH, S.B. YAO
"DIALOGUE : a Distributed-Processor Organization for a Database Machine" National Computer Conference, 1980, pp 243-253,.

WONG76 E. WONG, K. YOUSSEFI
"Decomposition - a Strategy for Query Processing" ACM Transactions on Data Base Systems, vol.1, no3, september 1976.

ANNEXES

Annexe 1

PARAMETRES MATERIELS

Ici sont fixées les valeurs des paramètres déclarés aux paragraphes 3 des chapitres III et IV :

V : le temps de comparaison de deux attributs = 10 microsecondes,

I : le temps de transfert d'un tuple dans une page = 250 microsecondes, en se basant sur une taille de page de 16K octets et une taille de tuple de 160 octets,

t : le nombre de tuples dans une page = 100,

Ct : coefficient d'occupation d'un attribut de semi-jointure =0.1, ce qui fixe une longueur d'attribut à 16 octets,

F, F' : taux d'efficacité de la mémoire cache, F=0.8, F'=0.3,

Tdc : le temps de transfert d'une page du disque vers la mémoire cache = 40 millisecondes,

Tcp : le temps de transfert d'une page de la mémoire cache vers un processeur = 16 millisecondes, basé sur un débit de bus moyen d'un mega octets par seconde,

M : le coût de traitement d'un message incluant l'envoi, le transfert et la reception est supposé être 15 millisecondes,

CL : suivant les coûts précédants, le coût moyen de lecture d'une page par un processeur est 54 millisecondes,

CE : le coût moyen d'écriture d'une page par un processeur est alors 74 millisecondes.

Annexe 2

JEUX D'ESSAI DU PROCESSEUR DE JOINTURE ET TRI

La base de données "JEU D'ESSAI" comporte les relations EMPLOYE, SERVICE et VOITURE, définies ainsi :

EMPLOYE

nom	age	service	salaire
dupont	40	vente	5000
dupond	25	achat	8500,50
durand	30	vente	8000
toto	30		2000
dupuy	60	direction	9000
randu	25	vente	8000

SERVICE

service	directeur	salaire
vente	pierre	8000
achat	paul	8500,50
personnel	jacques	9999,99
	jean	3000
direction	directeur	9000

VOITURE

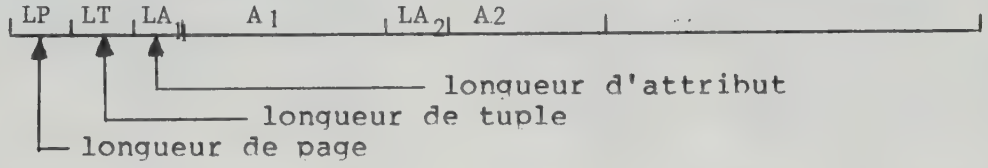
no	marque	type	prix
15	citroen	GS	45000
20	mercedes	200	80000
10	renault	18	40500,80
10	renault	5TS	29000
35	abarth	bianci	30000
10	renault	5TL	29000,50
15	citroen	CX	50000
10	renault	5TS	29000
35	abarth	bianci	30000
10	renault	5TS	29000

REMARQUES

Le programme du PJT est réalisé en PASCAL sous le système MULTICS (HB68). (voir Chapitre V).

Les relations opérandes et résultat sont gérées en mémoire cache sous forme de fichiers MULTICS, et sont lues dans des messages.

Le PJT et le Processeur de la Mémoire Cache échangent des pages de longueur 128 octets pour la simulation. Le format d'une page est le suivant :



LE PROGRAMME TESTPJT

- en mode conversationnel, il permet d'entrer les paramètres d'une opération JOINDRE, TRIER ou DEDOUBLER,
- il appelle le programme demandé. La trace montre les appels au processeur de la mémoire cache.

TESTS DU PJT

1) l'opération JOINDRE (page 175)

jointure des relations EMPLOYE et SERVICE sur l'attribut service et salaire, ce qui permet de connaitre les employés gagnant le même salaire que leur chef

2) l'opération DEDOUBLER (page 178)

élimination des tuples dupliqués de la relation VOITURE,

3) l'opération TRIER (page 180)

trie la relation VOITURE dédoublée en ordre ascendant sur la marque et en ordre descendant sur le prix.

la relation EMPLOYE

```
***  PROCESSEUR DE GESTION DE LA MEMOIRE CACHE  ***

lecture de la relation permanente  7

longueur du message transmis : 136
LONGUEUR DU MESSAGE :  136
t* :  1   lt* : 21
                        a* :  1   la* :  6     dupont
                        a  :  2   la  :  2     40
                        a  :  3   la  :  5     vente
                        a  :  4   la  :  4     5000
t :  2   lt : 24
                        a  :  1   la  :  6     dupond
                        a  :  2   la  :  2     25
                        a  :  3   la  :  5     achat
                        a  :  4   la  :  7     8500,50
t :  3   lt : 21
                        a  :  1   la  :  6     durand
                        a  :  2   la  :  2     30
                        a  :  3   la  :  5     vente
                        a  :  4   la  :  4     8000
t :  4   lt : 14
                        a  :  1   la  :  4     toto
                        a  :  2   la  :  2     30
                        a  :  3   la  :  0
                        a  :  4   la  :  4     2000
t :  5   lt : 24
                        a  :  1   la  :  5     dupuy
                        a  :  2   la  :  2     60
                        a  :  3   la  :  9     direction
                        a  :  4   la  :  4     9000
t :  6   lt : 20
                        a  :  1   la  :  5     randu
                        a  :  2   la  :  2     25
                        a  :  3   la  :  5     vente
                        a  :  4   la  :  4     8000
```

* { t : tuple
  { lt : longueur tuple
  { a : attribut
  { la : longueur attribut

la relation SERVICE

```
***  PROCESSEUR DE GESTION DE LA MEMOIRE CACHE  ***

lecture de la relation permanente  8

longueur du message transmis : 109
LONGUEUR DU MESSAGE :  109
t :  1   lt : 18
                        a :  1   la :  5     vente
                        a :  2   la :  6     pierre
                        a :  3   la :  4     8000
t :  2   lt : 19
                        a :  1   la :  5     achat
                        a :  2   la :  4     paul
                        a :  3   la :  7     8500,50
t :  3   lt : 26
                        a :  1   la :  9     personnel
                        a :  2   la :  7     jacques
                        a :  3   la :  7     9999,99
t :  4   lt : 11
                        a :  1   la :  0
                        a :  2   la :  4     jean
                        a :  3   la :  4     3000
t :  5   lt : 25
                        a :  1   la :  9     direction
                        a :  2   la :  9     directeur
                        a :  3   la :  4     9000
```

```
testpjt
          PROCESSEUR DE JOINTURE ET TRI
      trace demandee : 0 (pas de trace) ou 1 (trace) ?
?1
   operation demandee : taper 1 (JOINDRE) ou 2 (TRIER) ou 3 (DEDOUBLER
?1
    PROGRAMME JOINDRE
      numero de la 1ere relation ?
?7
      numero de la 2eme relation ?
?8
      numero de la relation resultat ?
?9
      nombre de predicats d equi-jointure ?
?2
predicat numero 1
      relation 1
      numero d'attribut ?
?3
      type de l'attribut ?
      taper 1 (entier), 2 (reel) ou 3 (texte)
?3
      relation 2
      numero d'attribut ?
?1
      type de l'attribut ?
      taper 1 (entier), 2 (reel) ou 3 (texte)
?3
predicat numero 2
      relation 1
      numero d'attribut ?
?4
      type de l'attribut ?
      taper 1 (entier), 2 (reel) ou 3 (texte)
?2
      relation 2
      numero d'attribut ?
?3
      type de l'attribut ?
      taper 1 (entier), 2 (reel) ou 3 (texte)
?2
      nombre d'attributs resultats ?
?4
      attribut resultat numero 1
      numero relatif de relation source (1 ou 2)?
?1
      numero d'attribut de la relation source ?
?1
      attribut resultat numero 2
      numero relatif de relation source (1 ou 2)?
?1
      numero d'attribut de la relation source ?
?2
      attribut resultat numero 3
      numero relatif de relation source (1 ou 2)?
?2
      numero d'attribut de la relation source ?
?2
      attribut resultat numero 4
      numero relatif de relation source (1 ou 2)?
?2
      numero d'attribut de la relation source ?
?3
```

Execution

```
***  PROCESSEUR DE JOINTURE ET TRI  ***
     jointure des relations  7  et   8  dans la relation  9
          attributs resultats
               relation  7, attribut de position  1
               relation  7, attribut de position  2
               relation  8, attribut de position  2
               relation  8, attribut de position  3
          predicat de jointure
              attribut de position  3 = attribut de position  1
 et           attribut de position  4 = attribut de position  3

     algorithme par produit
positionnement dans la relation permanente  9 : page  0
positionnement dans la relation permanente  7 : page  1
lecture de la relation permanente  7 : page  1
positionnement dans la relation permanente  8 : page  1
lecture de la relation permanente  8 : page  1
jointure de deux pages
lecture de la relation permanente  8 : fin de fichier
lecture de la relation permanente  7 : page  2
positionnement dans la relation permanente  8 : page  1
lecture de la relation permanente  8 : page  1
jointure de deux pages

lecture de la relation permanente  8 : fin de fichier
lecture de la relation permanente  7 : fin de fichier
ecriture dans la relation permanente  9 : page  1
     fin de la jointure
        OPERATION TERMINEE
r  17:27 Mon 30/11/81


     ***  PROCESSEUR DE GESTION DE LA MEMOIRE CACHE  ***

     lecture de la relation permanente  9

     longueur du message transmis :  98
     LONGUEUR DU MESSAGE :   98
     t :  1   lt : 23
                          a :  1   la :  6     dupond
                          a :  2   la :  2     25
                          a :  3   la :  4     paul
                          a :  4   la :  7     8500,50
     t :  2   lt : 22
                          a :  1   la :  6     durand
                          a :  2   la :  2     30
                          a :  3   la :  6     pierre
                          a :  4   la :  4     8000
     t :  3   lt : 24
                          a :  1   la :  5     dupuy
                          a :  2   la :  2     60
                          a :  3   la :  9     directeur
                          a :  4   la :  4     9000
     t :  4   lt : 21
                          a :  1   la :  5     randu
                          a :  2   la :  2     25
                          a :  3   la :  6     pierre
                          a :  4   la :  4     8000
```

la relation VOITURE

*** PROCESSEUR DE GESTION DE LA MEMOIRE CACHE ***

lecture de la relation permanente 5

longueur du message transmis : 238
LONGUEUR DU MESSAGE : 238

t : 1 lt : 20

	a : 1	la : 2	15
	a : 2	la : 7	citroen
	a : 3	la : 2	GS
	a : 4	la : 5	45000

t : 2 lt : 22

a : 1	la : 2	20
a : 2	la : 8	mercedes
a : 3	la : 3	200
a : 4	la : 5	80000

t : 3 lt : 23

a : 1	la : 2	10
a : 2	la : 7	renault
a : 3	la : 2	18
a : 4	la : 8	40500,80

t : 4 lt : 21

a : 1	la : 2	10
a : 2	la : 7	renault
a : 3	la : 3	5TS
a : 4	la : 5	29000

t : 5 lt : 23

a : 1	la : 2	35
a : 2	la : 6	abarth
a : 3	la : 6	bianci
a : 4	la : 5	30000

t : 6 lt : 24

a : 1	la : 2	10
a : 2	la : 7	renault
a : 3	la : 3	5TL
a : 4	la : 8	29000,50

t : 7 lt : 20

a : 1	la : 2	15
a : 2	la : 7	citroen
a : 3	la : 2	CX
a : 4	la : 5	50000

t : 8 lt : 21

a : 1	la : 2	10
a : 2	la : 7	renault
a : 3	la : 3	5TS
a : 4	la : 5	29000

t : 9 lt : 23

a : 1	la : 2	35
a : 2	la : 6	abarth
a : 3	la : 6	bianci
a : 4	la : 5	30000

t : 10 lt : 21

a : 1	la : 2	10
a : 2	la : 7	renault
a : 3	la : 3	5TS
a : 4	la : 5	29000

```
testpjt
          PROCESSEUR DE JOINTURE ET TRI
      trace demandee : 0 (pas de trace) ou 1 (trace) ?
?1
   operation demandee : taper 1 (JOINDRE) ou 2 (TRIER) ou 3 (DEDOUBLER)
?3
          PROGRAMME DEDOUBLER
      numero de la relation source ?
?5
      numero de la relation cible
?6
      nombre d'attributs par tuple
?4
         attribut numero   1
         type de l'attribut : 1(entier), 2(reel) ou 3(texte) ?
?1
         attribut numero   2
         type de l'attribut : 1(entier), 2(reel) ou 3(texte) ?
?3
         attribut numero   3
         type de l'attribut : 1(entier), 2(reel) ou 3(texte) ?
?3
         attribut numero   4
         type de l'attribut : 1(entier), 2(reel) ou 3(texte) ?
?2
```

Exécution

```
FUNCTION DEDOUBLER
creation de la relation de travail numero 91
***  PROCESSEUR DE JOINTURE ET TRI  ***

     tri de la relation  5  dans la relation 91

     critere
          position  1, ascendant
          position  2, ascendant
          position  3, ascendant
          position  4, ascendant


     fin du tri
positionnement dans la relation de travail 91 : page  1
lecture de la relation de travail 91 : page  1
lecture de la relation de travail 91 : page  2
ecriture dans la relation permanente  6 : page  1
lecture de la relation de travail 91 : fin de fichier
ecriture dans la relation permanente  6 : page  2
        OPERATION TERMINEE
r  17:48 Mon 30/11/81
```

```
***  PROCESSEUR DE GESTION DE LA MEMOIRE CACHE  ***

lecture de la relation permanente  6

longueur du message transmis : 167
LONGUEUR DU MESSAGE :  167
t :  1   lt : 23
                        a :  1   la :  2     10
                        a :  2   la :  7     renault
                        a :  3   la :  2     18
                        a :  4   la :  8     40500,80
t :  2   lt : 24
                        a :  1   la :  2     10
                        a :  2   la :  7     renault
                        a :  3   la :  3     5TL
                        a :  4   la :  8     29000,50
t :  3   lt : 21
                        a :  1   la :  2     10
                        a :  2   la :  7     renault
                        a :  3   la :  3     5TS
                        a :  4   la :  5     29000
t :  4   lt : 20
                        a :  1   la :  2     15
                        a :  2   la :  7     citroen
                        a :  3   la :  2     CX
                        a :  4   la :  5     50000
t :  5   lt : 20
                        a :  1   la :  2     15
                        a :  2   la :  7     citroen
                        a :  3   la :  2     GS
                        a :  4   la :  5     45000
t :  6   lt : 22
                        a :  1   la :  2     20
                        a :  2   la :  8     mercedes
                        a :  3   la :  3     200
                        a :  4   la :  5     80000
t :  7   lt : 23
                        a :  1   la :  2     35
                        a :  2   la :  6     abarth
                        a :  3   la :  6     bianci
                        a :  4   la :  5     30000
```

```
testpjt
          PROCESSEUR DE JOINTURE ET TRI
     trace demandee : 0 (pas de trace) ou 1 (trace) ?
?1
   operation demandee : taper 1 (JOINDRE) ou 2 (TRIER) ou 3 (DEDOUBLER
?2
          PROGRAMME TRIER
     numero de la relation source
?6
     numero de la relation cible
?9
     nombre de criteres de tri ?
?2
     critere de tri 1
     numero d'attribut ?
?2
     sens du tri ? 1 (ascendant) ou 0 (descendant)
?1
     type de l'attribut : 1 (entier), 2 (reel) ou 3(texte) ?
?3
     critere de tri 2
     numero d'attribut ?
?4
     sens du tri ? 1 (ascendant) ou 0 (descendant)
?0
     type de l'attribut : 1 (entier), 2 (reel) ou 3(texte) ?
?2
***  PROCESSEUR DE JOINTURE ET TRI  ***

     tri de la relation  6  dans la relation  9

     critere
          position  2, ascendant
          position  4, descendant

positionnement dans la relation permanente  6 : page  1
creation de la relation de travail numero 91
creation de la relation de travail numero 92
positionnement dans la relation de travail 91 : page  0
positionnement dans la relation de travail 92 : page  0
lecture de la relation permanente  6 : page  1
tri interne d'une page
ecriture dans la relation de travail 91 : page  1
lecture de la relation permanente  6 : page  2
tri interne d'une page
ecriture dans la relation de travail 92 : page  1
lecture de la relation permanente  6 : fin de fichier
fusion des relations 91 et 92 dans la relation  9
positionnement dans la relation de travail 91 : page  1
positionnement dans la relation de travail 92 : page  1
positionnement dans la relation permanente  9 : page  0
lecture de la relation de travail 91 : page  1
lecture de la relation de travail 92 : page  1
ecriture dans la relation permanente  9 : page  1
lecture de la relation de travail 91 : fin de fichier
ecriture dans la relation permanente  9 : page  2
lecture de la relation de travail 92 : fin de fichier
destruction de la relation de travail numero 91
destruction de la relation de travail numero 92
     fin du tri
        OPERATION TERMINEE
r  17:53 Mon 30/11/81
```

```
***  PROCESSEUR DE GESTION DE LA MEMOIRE CACHE  ***

lecture de la relation permanente  9

longueur du message transmis : 167
LONGUEUR DU MESSAGE :  167
t :  1   lt : 23
                        a :  1   la :  2     35
                        a :  2   la :  6     abarth
                        a :  3   la :  6     bianci
                        a :  4   la :  5     30000
t :  2   lt : 20
                        a :  1   la :  2     15
                        a :  2   la :  7     citroen
                        a :  3   la :  2     CX
                        a :  4   la :  5     50000
t :  3   lt : 20
                        a :  1   la :  2     15
                        a :  2   la :  7     citroen
                        a :  3   la :  2     GS
                        a :  4   la :  5     45000
t :  4   lt : 22
                        a :  1   la :  2     20
                        a :  2   la :  8     mercedes
                        a :  3   la :  3     200
                        a :  4   la :  5     80000
t :  5   lt : 23
                        a :  1   la :  2     10
                        a :  2   la :  7     renault
                        a :  3   la :  2     18
                        a :  4   la :  8     40500,80
t :  6   lt : 24
                        a :  1   la :  2     10
                        a :  2   la :  7     renault
                        a :  3   la :  3     5TL
                        a :  4   la :  8     29000,50
t :  7   lt : 21
                        a :  1   la :  2     10
                        a :  2   la :  7     renault
                        a :  3   la :  3     5TS
                        a :  4   la :  5     29000
```

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique

ISBN 2 - 7261 - 0279 - 4